

# EXPLORE COMPUTING with the TRS-80

and COMMON SENSE 🐛



by Richard V. Andree & Josephine P. Andree



THIS IS A DUPLICATED COPY OF A MANUSCRIPT TO BE PUBLISHED IN 1981 BY PRENTICE-HALL PUBLISHING COMPANY. IT IS FULLY COVERED BY U.S. AND FOREIGN COPYRIGHTS AND SHOULD NOT BE REPRODUCED IN WHOLE OR PART WITHOUT THE EXPRESS WRITTEN PERMISSION OF THE AUTHORS AND THE COPYRIGHT HOLDER.

© 1980 BY RICHARD V. ANDREE  
DUPLICATED IN NORMAN, OKLAHOMA, U. S. A.

YOUR AUTHORS WISH TO EXPRESS OUR SINCERE THANKS TO THE MANY TEACHERS WHO HAVE HELPED IN THE DEVELOPMENT OF THIS BOOK, AND TO ENCOURAGE YOU TO CORRESPOND WITH US CONCERNING YOUR CORRECTIONS AND SUGGESTIONS ON WHAT YOU WOULD CONSIDER PARTICULARLY HELPFUL IN AN INSTRUCTOR'S MANUAL.

OUR "COVER GIRL" IS SUZANNE M. ANDREE, AN ELEMENTARY MATHEMATICS-SCIENCE TEACHER WHO USES COMPUTERS IN HER



## PART I

## TO THE READER

The first 100 pages carefully lead you from "watching the computer count fast" to where you have enough skill and confidence to write your own computer programs for creating random art, solving equations, or exploring microRESEARCH problems that interest you.

- If you are a complete novice, start with the introduction.
- If you are a beginner, start with Lesson 1. Work as many practice problems as you need.
- If you are a bit more sophisticated, read the summary at the end of Lesson 1, work a few problems from Practice Session 1. Then continue with Lesson 2.
- If you can already program in BASIC, read the summaries at the end of Lessons 1 & 2, work a few problems, and continue with Lesson 3.
- Lesson 4 introduces graphics, computer assisted art and some of the special goodies available on the TRS-80.
- Lesson 5 is devoted to solving problems that are easy to understand, but which would be very hard to solve without the assistance of a computer.

## PART II

The first five Lessons will teach you most of what you need to know to use your computer in solving real problems.

The nature of Lessons 6 to 17 is different. These lessons are devoted to special topics you may or may not need to learn about, depending upon your own particular interests. Lessons 6 to 17 may be read in any order, or ignored entirely until needed. Consult the table of contents for topics covered.

The important thing is to HAVE FUN with your TRS-80. Remember, today's micro-computers are more versatile and more powerful than computers which cost 100 times as much, 20 years ago. Whether you purchased your TRS-80 as a playtoy or a serious tool, we hope you will soon be using it for both.

## TO THE INSTRUCTOR

This book is designed to be used by students for self-instruction (with or without a teacher's assistance). It has been successfully used by junior and senior high school students, secretaries, college students, 6th grade math classes, as well as a group of busy doctors and business people. Numerous examples are developed step by step to demonstrate the thinking involved. Since computing is a skill that must be learned by practice, a variety of problems are included. As with any acquired skill, a good coach can keep the neophyte interested and help develop good habits. Computing can and should be FUN. Students are fascinated by computers and will be responsive to your teaching.

Most readers are familiar with whole-number arithmetic, but may not have investigated the many tantalizing arithmetic puzzles that are easy to understand but hard to solve, unless a computer is available. A number of our problems come from the area of Number Theory. Problems like

1. Find squares like  $(11)^2 = 121$  or  $(264)^2 = 69696$  whose digits read the same forward as backward (palindromic squares),
2. Find squares such as  $(35853)^2 = 1285437609$  in which all ten digits are present in the square.

are not difficult to program once the rudiments are learned (Lessons 1, 2,3). Each reader is encouraged to use common sense to write more effective programs. Lessons 4,5,7,9,13,14, and 17 provide numerous interesting problems from non-mathematical areas. We hope each reader will discover new interests and power as his programming skills develop.

Readers are encouraged to develop orderly (top-down, structured) programming habits without being preached at. Your assistance is sought in helping students develop this vital skill.

Lessons 5 and 17 present 75 micro-research problems that are easy to understand. Micro-research problems demand thoughtful program design to produce results with a reasonable investment of computer time. All have been successfully programmed by students with no more than 9th grade mathematical skill. In many cases, the programs "develop" as do the examples given in the text, starting first with a brute-force attack which could require weeks of computer time, and then gradually, as the early results are available for examination, refinements in program design (early trimming of logic trees) suggest themselves. Eventually, many students devise programs that produce more results in 5 minutes than the original program could produce in five days.

Appreciation and understanding of the value of skillful computing techniques are developed as needed, through experience.



The material has been used with Models I, II, & III TRS-80 micro-computers to teach introductory computing concepts to several different groups, including students in grades 5 through 14, adult neophytes having no experience on computers and little math beyond the 8th grade, secretaries and secretaries-in-training, a group of busy doctors, bankers and business people, and casual adult and child visitors at a museum. Each contributed to the presentation contained herein. Every vital concept is presented more than once, in different environments, to aid the student's comprehension. The important thing is to be sure readers also actively program computers, not passively read the text. Passive reading, like a spectator sport, does not develop skills. Encourage students to really program various kinds of problems.

With the exception of Lesson 4 ART & GRAPHICS and a few special goodies available only on the TRS-80 Level II BASIC, most of the material presented here is applicable to any computer using BASIC.

The first five lessons form a unit. Lessons 6 through 17 may be taken up in any order or ignored, as individual interests suggest.

Our experience suggests having two or three students per computer produces better results than one person per computer--particularly in the early stages.

Your authors welcome criticism, suggestions and correspondence.

Richard V. Andree & Josephine P. Andree

The University of Oklahoma

Norman, Oklahoma 73019



### ACKNOWLEDGEMENTS

This book has evolved gradually over a three-year period of constant use, revision and modification. Hundreds of teachers and students from all over Oklahoma used the preliminary versions. Their suggestions and reactions helped us improve and revise the material. We express our gratitude to each teacher and to each student who participated.

Our very special thanks to:

Andrew L. Stout  
Gary Capps  
C. David Beatty

who did the photographic work for illustrations, cartoons, and chapter headings.

Suzanne M. Andree, our cover girl,  
is an elementary school teacher who uses TRS-80's in her teaching.

It is impossible to express individual appreciation to all involved, but it would be churlish to omit naming a few of those whose devoted assistance brought it all together.

Our sincere appreciation to:

Doug Mitchell  
Carolyn Thomson  
Mary Roland  
Nancy Dixon  
Patty Porter  
Tim Scovill  
Rosemary Dorman  
Bob Yarbrough  
Richard Odendahl  
Alice Shelton  
Anthony Tipton  
Karen Henry  
Michael Briggs  
Lana Pierce

each of whom has contributed far and above the call of duty.

We sincerely hope you will share their enthusiasm.



## CONTENTS

To the READER

To the INSTRUCTOR

ACKNOWLEDGEMENTS

### PART I

#### INTRODUCTION FOR THE NOVICE      1

If you have never used a computer before, this introduction was written especially to help you gain familiarity with the keyboard, the screen and the use of **BREAK**, **CLEAR**, and **ENTER** keys. Your computer will do arithmetic and display messages. The main purpose of the introduction is to lift you from neophyte to beginner.

#### LESSON 1      MAKE YOUR TRS-80 WORK      6

Beginners start here. A four-step program shows off your computer's speed as well as its versatility. Program instructions, PRINT variations, and additional keyboard commands are introduced.

#### LESSON 2      CREATING TABLES      17

Learn by programming the computer to create tables of squares, cubes, etc., for varying ranges and different step sizes. You are still not an expert, but you are well started.

#### LESSON 3      COMPUTER-ASSISTED SOLUTIONS      33

You and your computer will solve problems that would be difficult, if not impossible, to solve without a computer. Nasty equations are easy to understand and to solve with the help of a simple computer program.



LESSON 4      ART & GRAPHICS      47

Although this lesson is mostly fun, the ideas developed help design computer games and present graphical material in a dynamic format that is easy to understand. Lots of room for fun and self-expression here.

LESSON 5      micro-RESEARCH PROBLEMS      75

Surprising as it may seem, you already have enough computing-know-how to solve problems that would be too difficult to consider without computer assistance. A bit of eighth-grade mathematics, common sense, and your computer are all that is needed to investigate problems that would intimidate a college student who did not have access to a computer or calculator.

PART II      101

A review of the contents of the rest of the book--which may be taken up in any order desired, or ignored, whichever serves your needs best.

LESSON 6      TAPE CASSETTES      103

As you develop longer programs you will wish to save the program on magnetic tape using the cassette player that came with your computer. Tape cassettes are a standard way of trading programs with other computer owners. Cassettes are used for storing data and results as well as programs.

LESSON 7      GAMES & PLAYTOYS      109

Everyone loves to play computer games. Use your own ingenuity to create new game programs and to improve programs written by others.

LESSON 8      EDIT INSTRUCTIONS      123

A serious programmer needs to know more about the powerful EDIT instructions available on your computer. This is where you will find answers.

LESSON 9      SIMULATION      129

College courses are given on SIMULATION, but the basic philosophy involved is easy to understand and not difficult to program. This introduction removes much of the mystery and enables you to program your own simulations and computer games.



LESSON 10      TYPES OF VARIABLES      143

Single precision floating point	Double precision floating point
Integer variables (uses and limitations)	String variables
Subscripted variables	Arrays and matrices

LESSON 11      TIPS AND ERROR MESSAGES      150

Experience may be the best teacher, but you might as well profit from the experience of others.

Tips on using TRS-80 BASIC:

- Saving memory
- Speeding up program execution
- Error messages and what they tell you

LESSON 12      EXTENDED PRINT INSTRUCTIONS      159

You may never need more PRINT instructions than those introduced in Lessons 1, 3, 4. However, your microcomputer has several extra goodies ready to meet your needs if and when they arise.

LESSON 13      MORE GRAPHICS      165

Lesson 4 introduced the most commonly used graphics, but your TRS-80 has a number of convenient extras you may wish to investigate.

LESSON 14      STRING AND LOGICAL OPERATORS      169

If "word processing" interests you, you should investigate the TRS-80 STRING and LOGICAL operations more carefully. Two SENTENCE GENERATORS, A BUZZ-WORD PROGRAM, A ROBOT COUNSELOR and a fairly secure CIPHER PROGRAM, are included, as well as a more detailed examination of the LOGICAL OPERATORS: AND, OR, NOT.

LESSON 15      WHERE TO LOOK FOR ADDITIONAL INFORMATION      180

Tell "Greatest Martha" you would much rather have a subscription to one of these magazines than a box of handkerchiefs for your next birthday. You may get both.



LESSON 16      ADDITIONAL BASIC      184

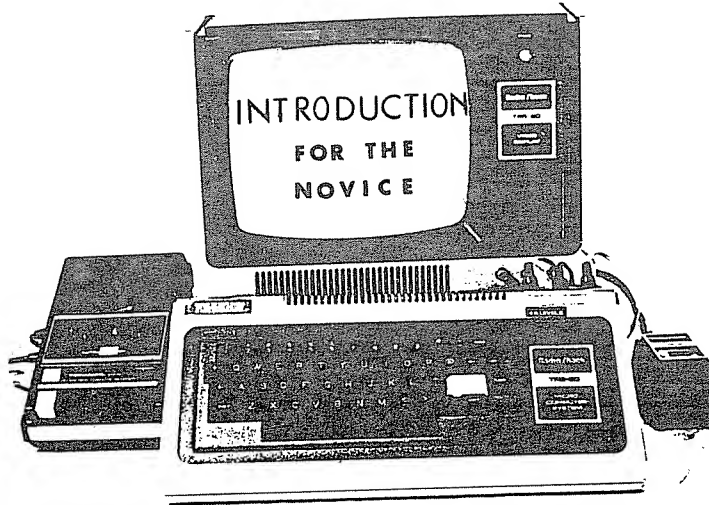
Additional BASIC instructions are available if and when you need them. Turn here if you need to do something we haven't discussed.

LESSON 17      MORE micro-RESEARCH PROBLEMS      188

Lesson 5 presented 17 micro-RESEARCH PROBLEMS you could explore with the computing expertise available then. Here we discuss programming practice, design and improvement of programs using easy-to-follow examples. Lesson 17 then completes our collection of seventy-five (75) choice micro-RESEARCH problems, each worthy of exploration, expansion, and investigation. This collection contains a wide variety of proposals including several arithmetical explorations, Haiku poetry, logical decisions, calendar problems, dart and target games, loan and interest programs, treasure hunt games, caricatures, a speech timer, puzzles, recursive functions, polygonal maps, medical emergency prompter, number theory, lattice problems, magic prime squares, graphs, monkeys at typewriters, amicable and sociable numbers, dance partners, and the notorious eight queens problem. They are here for your entertainment and education. Enjoy them.

A FINAL WORD from your authors      214

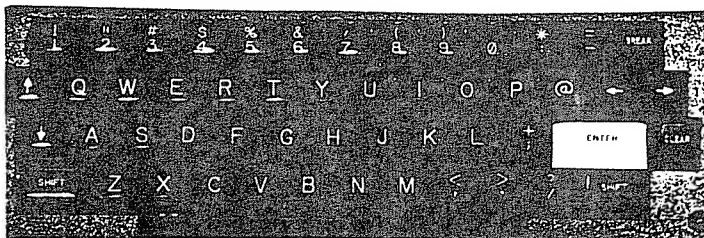
INDEX      215





We assume you have no knowledge of computing, but are interested in learning. The way to learn computing is to compute!

Just as in golf or bowling, it is more helpful to read about computing after you have tried your hand at the game enough to have a feel for its rudimentary swing. So let's begin.

Let's start by using the computer as if it were a pocket calculator. It is a waste of the computer's power, but it will help you become familiar with the keyboard. Have someone turn it on for you. Depress the white **ENTER** key a couple of times.



The keyboard is similar to a typewriter. If you wish to use a symbol like = or +   from the top row of the key, depress the **SHIFT**

key. The computer automatically types capital letters, as shown on the keys, so do not use the shift key unless you wish a symbol from the top half of a key.

Don't try to "read" this introduction—the way to learn computing is to compute, so wait until you have a TRS-80 at your fingertips before continuing.



Depress the following keys:

1. **BREAK**

(This interrupts the computer in case it was doing something.)

2. **ENTER**

(This is the key you will use to send your typed message to the computer.)

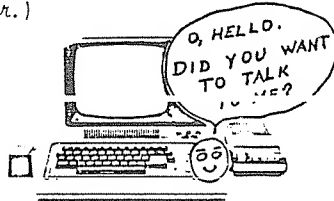
3. Now type: `PRINT 3+5+9-2` **ENTER**

The computer should respond with the sum 15.

Your CRT-display will now show:

```

READY
>PRINT 3+5+9-2
15
READY
>-
  
```



The > symbol indicates the computer is ready to receive your typed instruction.

The - shows where the next typed character will appear.

Should you wish to clear the screen, depress the **CLEAR** button, but there is no need to do so unless you wish it cleared. You may also type `CLS` **ENTER** to Clear the Screen.

What will the answer be if you type ?

`PRINT 4*9 - 2` **ENTER**

Most computers use the symbols  
\* for multiply and / for divide.

Try it and see.

Forecast the result of each of the following, and then try them out on your TRS-80.

`PRINT 4/3` **ENTER**

`PRINT 9*5` **ENTER**

`PRINT 360*18` **ENTER**

`PRINT 1/2 + 0.6` **ENTER**

Note 0 is zero on the top row.  
O is the letter "oh" which is on the second row. Your computer will be unhappy if you confuse them.  
Try it once and see for yourself.




Let's try some hard problems

`PRINT 774*817` **ENTER**


`PRINT 79276/511017` **ENTER**

`PRINT 317*1.41421 - 246*3.14159` **ENTER**

If you type `774*817` **ENTER** without `PRINT`, the computer will do the arithmetic but will not print or store the result.


The  key is another important key for those of us who occasionally mistype. It will backspace and erase one character at a time. If you hold down the  key while you depress  the entire line you typed will be erased. Try it and see for yourself.


Type


PRINT "HELLO " 

*Note the " " marks.  
What happened?*

Type

PRINT "SALUTATIONS " 

PRINT "MY NAME IS your name " 


Now let's try something different. This time we shall put a "line number" in front of each statement. First type 

Then type:

```
100 PRINT "HELLO " 
110 PRINT "SALUTATIONS " 
120 GOTO 100 
```


That isn't progress, or is it? Well, nothing happened. Did it? Yes, it did, but it happened inside the computer, where we can't see it, not on the screen.



Type


LIST 

*The computer should list the three statements you typed in. Check them to be sure they were properly entered with a space after each number and " " in the proper places.*

Next type

RUN 

After a bit, depress the  key to get control again. Depress the  button.

Type LIST 

and the computer should respond by displaying:

```
100 PRINT "HELLO "
110 PRINT "SALUTATIONS "
120 GOTO 100
READY
>_
```

It is perfectly reasonable to change these instructions by retyping them.



Type

100 PRINT "HELLO your name";  Note the " " and the ;.

110  This erases the instruction in 110  
as you will see when you type LIST .

Now type

LIST

The screen should show

100 PRINT "HELLO your name";

120 GOTO 100

Note that instruction 100 was changed and instruction 110 was deleted, but instruction 120 is unchanged.

Now type

RUN

Let us write another program. Begin by depressing  to get the computer's attention. Then type:

NEW

100 X = 1980

110 Y = 365

120 Z = X\*Y

130 PRINT X; Y, Z

140 GOTO 100

RUN

The above program sets X=1980, then sets Y=365, then forms the product of the values of X and Y and stores it in a location called Z. Instruction 130 then prints out the values of the numbers stored in locations X, Y, and Z. Instruction 140 then sends the computer back to instruction 100, which repeats the entire process.

Depress the  key to stop the computer.

SUMMARY OF INTRODUCTION

*Let's see what you have learned thus far...*

To get the computer's attention, depress **BREAK** key.

Once you get the computer's attention, which it indicates by displaying

```

READY
>_

```

on the CRT (TV screen), you can work arithmetic problems involving +, -, \*, / (add, subtract, multiply, divide) by typing PRINT followed by the arithmetic problem.

When the problem is displayed the way you wish it, depress **ENTER** to send it to the computer. Remember, nothing goes to the computer until you depress **ENTER**.

Oh, yes. If you make an error in typing, just depress the **←** key to backspace and erase. You can do it anytime before you depress the **ENTER** key.

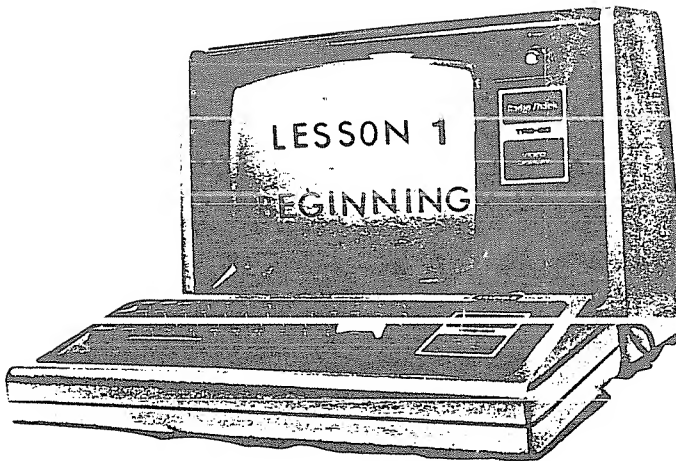
To clear the screen, depress the **CLEAR** key, followed by **ENTER** if the >\_ symbol is absent, or type CLS **ENTER**.

You need to be careful about confusing zero, 0, and the letter oh, O. Also, 1 and L are distinct.

You have written a set of instructions to the computer, and the computer has followed these instructions.

Now that you no longer fear the TRS-80, let's begin our lessons.

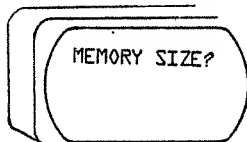




So you don't know anything about computers or computing. Well, this won't make you an expert, but let's see what we can do.

Someone has connected and assembled the TRS-80 and turned it on for you. The screen says something on it. I don't know what because I don't know what was done last.

If your TRS-80 has just been turned on, the screen may display



*The TRS-80 can set aside memory for programs written in languages other than BASIC, but we shall not do so.*

If so, just depress the white **ENTER** key. The screen will then display an additional message, which should end with



The word **READY** is your clue that the computer is ready to accept instructions from you.

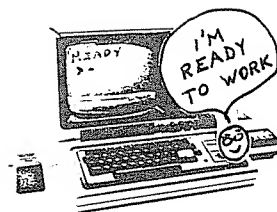
To clear up everything...

Depress the **BREAK** key (upper right).  
The computer will add

```

READY
>_

```



below whatever was on the screen. Now you have its attention. The **>\_** symbol below **READY** means the TRS-80 is waiting for you to type in an instruction.

Type **NEW** and depress the white **ENTER** key.

Everything on the screen disappears—also any program inside disappears too—the slate is wiped clean for you, and the computer displays

```

READY
>_

```

NOTE: Nothing goes from the screen to the computer until you depress the **ENTER** key.

Now type


**210 X = 1** **ENTER**

The screen should now show

```

READY
> 210 X=1
>_

```

If you mistype, just use the  button to backspace and erase--or depress **BREAK** and start the entire line over.

The symbol **>\_** indicates the TRS-80 is ready to accept another instruction, so type

**220 PRINT X;** **ENTER**

Don't forget the ;

The screen now shows

```

READY
> 210 X=1
> 220 PRINT X;
>_

```



Continuing, type

```
230 X = X + 1 
240 GOTO 220 
```

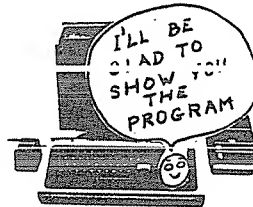
*Note that GOTO is one word.*

To make sure the program you typed is really there, type

```
LIST 
```

Your program will then be listed below whatever is already on the screen.

```
READY
>210 X=1
>220 PRINT X;
>230 X=X+1
>240 GOTO 220
>LIST
210 X=1
220 PRINT X;
230 X=X+1
240 GOTO 220
>_
```



Note the absence of the > symbols on the LISTed program, indicating the computer typed it to you.

Now you are ready to run your first numerical program.

Type RUN  and watch what happens.

You probably have a Level II BASIC, but if not, read the right-hand column instead of the left-hand column below.

#### Level II BASIC

If you hold down  and depress  , the program will halt until you depress another key (without  ). Try it a few times and see.

#### Level I BASIC

If you depress the  key on left edge of the keyboard, the program will halt whatever it is doing, as long as you keep the  key depressed, but will continue whenever you let it up. Try it a few times and see.

In either case, if you wish an extended halt, depress the  key. (Do so now.) Not only does the TRS-80 stop, but it even tells you which instruction it was executing when you depressed  . If you want the program to continue, simply type: CONT  and it will continue.

Note that in Level I BASIC, if the program is typing a number, say 425 or 438, and runs out of line, it prints part of the number at the end of one line and the rest of the number at the beginning of the next line. The output is difficult to read, but the data is printed quickly and the screen displays a lot of data at once.

Level II BASIC is a bit more sophisticated and will leave the end of the line blank, rather than break a number, but it still changes the column spacing as the number of digits in the number changes in order to get as much output on a line as is convenient. The ; at the end of

```
220 PRINT X;
```

is what controls the "packed spacing" of this output.

Depress **BREAK**

Type RUN **ENTER**

to see what is meant.

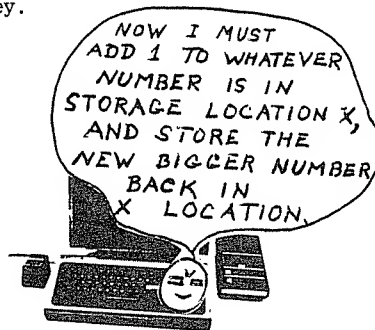
Let's look at our program again.

Depress the **BREAK** key and the **CLEAR** key.

Then type LIST **ENTER**

The computer should respond.

```
>LIST
210 X=1
220 PRINT X;
230 X=X+1
240 GOTO 220
READY
>
```



Instruction 230  $X=X+1$  looks odd. If it were a mathematical equation, it would have no solution. It is not a mathematical equation; it is an instruction to a computer.

$X=X+1$  This instructs the computer to "Take the number in the storage location called X, add 1 to it, store the new result back into the storage location called X."

$X=X+1$  could be more reasonably written as  $X=X+1$ . However, it is almost universal computer practice to use = rather than + , probably because old fashioned typewriter keyboards contain = not + .

Let's change instruction 220 PRINT X; by changing the semicolon to a comma. To do this, we simply type

```
220 PRINT X, ENTER
```

If you now type

```
LIST ENTER
```

the new program will be displayed below the LIST instruction. Please do so before continuing. Then type RUN **ENTER** .

Note that now our program prints out its values in four nice columns.

Isn't it impressive?

You may wonder what would happen if you did not have either the comma or the semicolon after PRINT X. If so, experiment a bit and find out.

Depress **BREAK** to get the computer's attention.

Type LIST **ENTER**

Type 220 PRINT X **ENTER**

Type LIST **ENTER**

Type PRINT **ENTER**

Depress **BREAK** to get the computer's attention.

Do some other experiments on your own. For example, change 230 X=X+1 to 230 X=X+.5 or 230 X=X+2 and see what happens when you type

RUN **ENTER**.

Remember, about the only thing you might do that would hurt the TRS-80 would be to drop part of it or spill a beverage over it. Otherwise, it is pretty rugged, as long as you don't open the case.



#### A Word of Warning:

It is always well to check that you have typed what you think you have typed before depressing the **ENTER** key... In most cases it is easy to correct an error by simply retyping that statement number and instruction, but if you should happen to have typed LLIST **ENTER** instead of LIST **ENTER** you may have a problem you will need to call your instructor to fix. LLIST is also a perfectly valid instruction—it tells the TRS-80 to list your program on the attached line printer—but if your TRS-80 has no line printer attached (and turned on), the computer will not accept another instruction until it executes the impossible instruction LLIST **ENTER**. To all appearances your computer will "lock-up." Your instructor can get you out of this difficulty without having to turn off the computer. If you are without an instructor, open the small door on the back left top side of the keyboard, and press the button inside.



If your TRS-80 has "keyboard bounce" (printing multiple letters on one keystroke), it means the contacts are dirty and no one inserted the "keyboard fix" program when the TRS-80 was turned on. Ask for help if you have excessive troubles with keyboard bounce. Each TRS-80 has a keyboard-fix tape and manual packed with it. If you do not have an instructor, consult the manual that accompanies the keyboard fix tape.

H A V E   F U N   ! ! !

#### SUMMARY OF LESSON 1

*Let's see what you have learned thus far...*

Depress **BREAK** key to get computer's attention if it is busy.

Type NEW **ENTER** to clear out everything and start over.

Number each instruction and put a space between the number and its instruction.

*(If you fail to number an instruction the TRS-80 will perform that instruction right then, but not store it for later use. Try typing PRINT X, Y, T, L **ENTER**.)*

You can erase errors by using the **←** key as a "backspace" if you have not depressed **ENTER**.

If you have depressed **ENTER** it's easy to change instructions, merely retype the number and new instruction. You may insert additional instructions between those already in use by using statement numbers that fall between those already used.

Holding down **SHIFT** while you depress **←** erases the entire line.

The symbol >- indicates the computer is ready to receive typed instructions from you.

If the >- symbol is not there, depress **ENTER** or **BREAK** **ENTER** .

In the PRINT instruction, the use of a semicolon ; between or after variables will give a "packed format", while the use of a comma, produces four columns. No symbol gives one output per line (wasteful).

You may halt the output (temporarily) by depressing the **↑** if you have a Level I. If you have a Level II, hold down **SHIFT** and depress **↓** . Depress any key except **SHIFT** or **BREAK** to continue.

LIST **ENTER** produces a listing of the program currently in storage.

Among your output you may discover some strange-looking numbers, such as 1.23456E+08 or 2.22222E-05. The computer displays only six significant decimal digits, and shows the magnitude by appending an E+nn or E-nn to indicate the power of ten by which the preceding fraction should be multiplied.

Thus

1.23456E+08 =  $1.23456 \times 10^{+08}$  = 123456000. *(But recall this may be in error by  $\pm 5$  in the seventh place, that is, by  $\pm 500$ .)*

2.22222E-05 =  $2.22222 \times 10^{-5}$  = .000022222 *( $\pm .00000000005$ )*

3.45678E+11 = 345678000000

-4.68024E-07 = -.000000468024

This convenient notation, a cousin of the so-called "scientific notation," is used on most modern computing systems, including many hand-held calculators.

PRINT X,Y;Z;T will print values of X Y Z T , spaced as shown.

**CLEAR** or **C** **L** **S** **ENTER** will clear the screen.

**SHIFT** **→** will produce double size letters on the screen (see problem 13).

Usually **CLEAR** is depressed just before using **SHIFT** **→** .

## PRACTICE SESSION 1

Problems 1 to 6 make use of the program discussed in the text. Start by typing it in, LISTing the program and RUNning it to check that it is functioning properly before you start the problem set.

depress **BREAK** key, then type:

```

NEW ENTER
210 X = 1 ENTER
220 PRINT X; ENTER
230 X = X + 1 ENTER
240 GOTO 220 ENTER

```

LIST **ENTER**

RUN **ENTER**

**BREAK**

CONT **ENTER**

**BREAK**



Now you are ready to start the practice set.

1. Type in the program (using 220 PRINT X;) first given in this lesson. Run it a few times with 230 X=X+1 replaced by each of the following in turn.

```

230 X=X+.5 ENTER
or 230 X=X+ 5 ENTER
or 230 X=X+ 2 ENTER
or 230 X=X+.1 ENTER

```

Type LIST **ENTER**

RUN **ENTER**

each time you change instruction 230. Note any peculiar behavior and think about it a bit. Discuss it with your instructor or a fellow computer buff.



2. In the program above, replace 210 X=1 with 210 X=7 and rerun the program.

3. Forecast the output of the program.

```
210 X=1
220 PRINT X,
230 X=X+X
240 GOTO 220
```

Try it out and test your forecast. What about column spacing? The error message `POV ERROR IN 230` indicates that the arithmetic operation in instruction 230 produced a result too large for the computer to handle—namely larger than  $10^{30}$ —which is a number larger than the total number of atoms in the solar system.

4. What output would you expect the program of problem 3 to produce if 240 GOTO 220 were replaced with 240 GOTO 210?

Try it out and verify your forecast, or explain why the computer did not do as you forecast.

What would happen if you used 240 GOTO 230?  
Be sure you understand these points.

5. Let's try another program. Depress `BREAK` if needed. Then type  
NEW `ENTER`.

If you know that most computing systems use \* to indicate multiplication, you should be able to forecast what the following program will produce.

```
210 X=2
220 PRINT X,
230 X=X*X
240 GOTO 220
```

Make a forecast. Then run the program. If the results are unexpected, reread problem 3.

6. Replace instruction 220 PRINT X, in problem 5 with 220 PRINT X, X+5 and see what happens. Before running the program, try to forecast the output of the first 5 lines.
7. Try some experiments on your own. Don't be afraid—you are not apt to harm the TRS-80 computer providing you don't eat or drink in the computer room and don't move or bump it while it is connected. Be sure to depress `BREAK` and type NEW `ENTER` before you start a new program.

*The new instructions used in problems 8,9,10 will be discussed in detail later. Now all you need to know is that  
IF RND(0)<.05 THEN CLS is pronounced "If 'random number'  
is less than .05 then clear the screen."*

8. Let's try another program. Depress **BREAK** if needed. Then type  
NEW **ENTER**.

Level II BASIC

```
10 PRINT @ RND(1000), "HI type your name"; ENTER
15 IF RND(0)<.05 THEN CLS ENTER
20 GOTO 10 ENTER
RUN ENTER
```

Level I BASIC

If you have a  
Level I BASIC re-  
place **@** with  
**A** **T** and it  
will behave  
similarly.

9. Experiment a bit by changing instruction 15 IF RND(0) < .05 THEN  
CLS to one of those suggested below.
- ```
15 IF RND(0) < .01 THEN CLS
or 15 IF RND(0) < .1 THEN CLS
or 15 IF RND(0) < .5 THEN CLS
or 15 CLS
```
- [Don't forget to depress **ENTER** each time you change an instruction.]
10. Try combining the program in the text with that of problem 8, with instruction 20 omitted, giving:
- ```
10 PRINT @ RND(1000), "HI type your name";
15 IF RND(0) < .05 THEN CLS (Type 20 ENTER to delete instr. 20.)
210 X=1
220 PRINT X;
230 X=X+1
240 GOTO 220
```
11. Change the instruction 240 GOTO 220 to 240 GOTO 10 in problem 10. Forecast the output before you RUN the program. If your forecast was not valid, try to find a why before reading problem 12.
12. Delete instruction 210 X=1 by typing 210 **ENTER**. When **>\_** appears, insert 5 X=1. Type LIST **ENTER** to check your new program. Before you RUN the program, try to forecast the result. Why is it different than that of problem 11 which used the same instructions, but placed differently in the program?
13. If you have a Level II BASIC, please try the following sometime when you have a program that does not contain CLS in the computer.
- Depress **BREAK** if needed.
- Depress **CLEAR**
- Hold down **SHIFT** and depress **→**
- Type LIST **ENTER**
- Notice that your program is listed in characters twice as large as before.
- Now type RUN **ENTER**
- The output is also double size. This is handy if you are using the computer with a class or group. Double size will last until **CLEAR**

key is depressed or until you execute a CLS instruction. To get the regular size back, depress BREAK and then CLEAR. If you fail to clear the screen before you depress SHIFT =>, your double-size letters and numbers obliterate half of the material on the screen to make room for the larger letters, but any material printed after that will be complete.

14. Have fun. Make your own variations.  
Forecast the result before you run each program.

Try NEW ENTER

210 X = 5 ENTER

220 PRINT X; X+5; X+10; X+15; ENTER

230 X = X+20 ENTER

240 GOTO 220 ENTER

Now investigate what happens if you omit the final ; in line 220. What happens if you change the semicolons to commas in line 220 (with and without the final comma)?

15. Can you write a program that will start with X = 1 and count by threes, with exactly four numbers on each line? There are several correct ways to write such a program, so when you are finished, compare your program with that of a friend who also has a computer.

16. Try the following program.

NEW ENTER

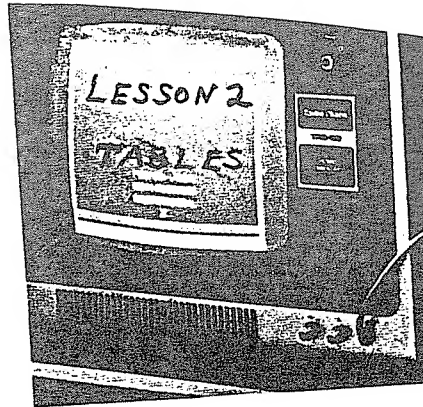
100 X = 2 ENTER

110 PRINT X, X\*X, X\*X\*X, X\*X\*X\*X ENTER

120 X = X+1 ENTER

130 GOTO 110 ENTER

RUN ENTER



In our first lesson, we learned to make the computer count, using various step sizes  
 to print in different formats, depending upon whether we used  
     PRINT X  
 or PRINT X;  
 or PRINT X,

We also learned the essential handling of the computer using **BREAK**, **ENTER** and **←** "backspace" keys and to type NEW **ENTER** to clear out old program and data, ready to accept new ones.

We rewrite the counting program using a slightly different philosophy below (for reasons you will soon appreciate.) We shall use B = beginning value, S = stepsize by which X will be increased, and F = final value, after which we wish the computer to stop.

```

NEW ENTER
2   B=1 ENTER
4   S=1 ENTER
6   F=100 ENTER
100 X = B ENTER
110 PRINT X; ENTER
120 X = X+S ENTER
130 IF X < F THEN 110 ENTER

```

Instruction 130 examines the current value of X and if X is less than F, sends the program back to instruction 110. If you feel at all insecure about this program, put it on the computer and RUN it before continuing.



The real advantage of this technique is that we can easily change the step size by changing instruction 4 to 4 S = .5

Indeed, we may rewrite the program slightly to input values of B, S, and F from the keyboard instead of assigning them in instructions 2, 4 and 6.

We do so next, using the instruction INPUT B,S,F . When the computer executes this instruction it will display a question mark on the TV screen and wait for you to type in three values, separated by commas. The values will be assigned, in the order typed, to variables B,S, and F. These letters were used as they are the initial letters of the phrases Beginning value, Step size, and Final value.

The revised program is:

```
NEW 
90 INPUT B,S,F 
100 X=B 
110 PRINT X; 
120 X=X+S 
130 IF X < F THEN 110 
140 PRINT 
150 PRINT "END OF TABLE WITH B,S,F="";B;S;F 
160 PRINT : PRINT 
170 GOTO 90 
RUN 
```

Again, note the use of instruction

```
130 IF X < F THEN 110.
```

As long as X is less than F, the program loops back to 110 PRINT X; but when  $X \geq F$ , the program continues with statements 140, 150, 160, and 170 which upspaces the printed matter, prints an END OF TABLE message, and then sends the computer back to 90 for a new set of input values.

Instruction 150 PRINT "END OF TABLE WITH B,S,F="";B;S;F is particularly interesting. It combines the printing of a message in quotes, " " with the printing of the values of the variables B,S and F, which must not be included within the quotes.

Try this program on your TRS-80 before continuing. Begin by typing RUN . When a ? appears on the screen, type 1, 1, 100 . Watch the output. When a ? appears at the bottom of the screen, type 2, .5, 10  or some B, S, F values of your own choice.

Let's write a program to create a table of values of

X       $X^2$        $X^3$        $X^4$

When our program is run, it will accept values B, S, F from the keyboard. It will set  $X=B$  and print out values of  $X$ ,  $X^2$ ,  $X^3$ ,  $X^4$  for that value of  $X$ . Then the instruction  $X=X+S$  will add  $S$  to the current value of  $X$  and print another set of  $X$ ,  $X^2$ ,  $X^3$ ,  $X^4$  for the new value before adding another  $S$  to the current  $X$ .

This loop will continue until  $X$  exceeds the value of  $F$  ( $X>F$ ) in which case the program will PRINT a blank line (160) followed by (170) an END OF TABLE message.

One way to do it is:

```

NEW  ENTER

100 PRINT 'PLEASE TYPE BEGINNING VALUE, STEP SIZE, FINAL VALUE ';ENTER
105 INPUT B,S,F ENTER
110 X=B ENTER
120 Y=X*X ENTER
130 PRINT X, Y, X*Y, Y*Y ENTER
140 X=X+S ENTER
150 IF X <= F THEN 120 ENTER
160 PRINT ENTER
170 PRINT 'END OF TABLE' ENTER

RUN  ENTER

```

Let us examine the program in detail.

The instruction

```
105 INPUT B,S,F
```

causes the computer to type a ? and wait for the user to type three values separated by commas and depress the ENTER key.

Instructions 100 and 105 can be combined into a single statement, but we shall not do so yet.

Instructions

```

110 X=B
120 Y=X*X
130 PRINT X, Y, X*Y, Y*Y

```

do just what you learned to expect in lesson one. Note that by using commas between output, we obtain four neat columns. Also note that the use of  $Y=X*X$  saves half of the multiplications that would be required by

$\text{PRINT } X, X*X, X*X*X, X*X*X*X$  since  $Y*X = X^3$  and  $Y*Y = X^4$ .

The instruction

```
140 X=X+S
```

merely increases the current value of  $X$  by the amount in  $S$ .

Next we use the branch instruction

```
150 IF X<=F THEN 120
```

This instruction compares the values stored in X and in F. If the value in X is less than or equal to ( $\leq$ ) the value in F, the program loops back to instruction 120. Otherwise the instruction which follows instruction 150 (namely 160 PRINT) is executed next.

```
160 PRINT
```

```
170 PRINT 'END OF TABLE'
```

Instruction 160 merely prints a blank line, while instruction 170 prints the message indicated in quotes. If after entering the program, you depress [CLEAR] type LIST [ENTER], the computer display will show

```
> LIST
100 PRINT "TYPE BEGINNING VALUE, STEP SIZE, FINAL VALUE";
105 INPUT B,S,F
110 X=B
120 Y=X*X
130 PRINT X, Y, X*Y, Y*Y
140 Y=X+S
150 IF X<= F THEN 120
160 PRINT
170 PRINT 'END OF TABLE'
READY
> -
```

If you now type RUN [ENTER], the computer will respond

```
> RUN
TYPE BEGINNING VALUE, STEP SIZE, FINAL VALUE?
```

If you respond with 1,1,5 [ENTER], the table of  $X$ ,  $X^2$ ,  $X^3$ ,  $X^4$  gives the below results:

```
1  1  1  1
2  4  8  16
3  9  27 81
4 16  64 256
5 25 125 625
END OF TABLE
READY
> -
```

If you type RUN [ENTER] again, the computer will again respond with  
TYPE BEGINNING VALUE, STEP SIZE, FINAL VALUE?

and if you type 1,.5,8 [ENTER], the screen will fill up and "scroll out the top" displaying

1.5	2.25	3.375	5.0625
2	4	8	16
2.5	6.25	15.625	39.0625
3	9	27	81
3.5	12.25	42.875	150.063
4	16	64	256
4.5	20.25	91.125	410.063
5	25	125	625
5.5	30.25	166.375	915.063
6	36	216	1296
6.5	42.25	274.625	1785.06
7	49	343	2401
7.5	56.25	421.875	3164.06
8	64	512	4096
END OF TABLE			
READY			
>_			

So, you see our computer works with fraction values as well as with whole numbers.

Note that  $(2.5)^4 = 39.0625$  which is correct, while for  $(3.5)^4$  the computer prints 150.063 instead of the correct value 150.0625. This is because, although the computer actually contains more than six digits of accuracy internally, it only displays six digits of accuracy. This will prove to be both a blessing and a curse, as we shall see.

If you are using a TRS-80 with Level II BASIC, it is quite feasible to obtain additional accuracy on it by declaring X and Y as DOUBLE PRECISION variables, but to do so now would only distract our learning procedure. Problems in "limited precision" and "rounding" also occur in double precision arithmetic--although not as frequently.

Let us type another set of values into our table. First, of course, depress CLEAR and type RUN ENTER. When the computer prompts you, type 0,.1,1 ENTER. You should obtain:

0	0	0	0
.1	.01	1E-03	1 E-04
.2	.04	8E-03	1.6E-03
.3	.09	.027	8.1E-03
.4	.16	.064	.0256
.5	.25	.125	.0625
.6	.36	.216	.1296
.7	.49	.343	.2401
.8	.64	.512	.4096
.9	.81	.729	.6561



Let's modify our program slightly by adding another instruction,  
106 CLS.  
To do so

Depress **BREAK** and **ENTER** (if necessary) to

obtain **READY** on your screen and then type

105 CLS **ENTER**

Next, depress **CLEAR** and type LIST **ENTER** to view the current program, which should start

```
100 PRINT "TYPE BEGINNING VALUE, STEP SIZE, FINAL VALUE";
```

```
105 INPUT B,S,F
```

```
106 CLS
```

```
110 X=B
```

```
...
```

```
...
```

```
...
```

with the new instruction 106 tucked properly between instruction 105 and 110.

Isn't that neat?

The instruction CLS is called clear screen. It clears the screen for you whenever it is executed. Try several B, S, F values of your own choice. Note that if you use B, S, F = 0, .1, 5, the table scrolls off the top before you can read it. You can stop this at any time you wish.

#### In Level II BASIC

Hold down **SHIFT** and depress **@** to stop the program. To continue the program, depress **@** without shift (or most any other key).

If you prefer, depress **BREAK** to stop the program. Type CONT **ENTER** to continue.

#### In Level I BASIC

Depress and hold down the **↑** key To continue the program, release the **↑** key.

It would be nice to have headings at the top of the column of  $X$ ,  $X^2$ ,  $X^3$ ,  $X^4$  we are printing. It is not hard. We add:

```
108 PRINT "X          SQUARE          CUBE          4-TH POWER" ENTER
```

So now our program reads:

```
100 PRINT "TYPE BEGINNING VALUE, STEP SIZE, FINAL VALUE";
105 INPUT B,S,F
106 CLS
108 PRINT "X          SQUARE          CUBE          4-TH POWER"
110 X = B
120 Y = X*X
130 PRINT X, Y, X*Y, Y*Y
140 X = X + S
150 IF X<=F THEN 120
160 PRINT
170 PRINT"  END OF TABLE"
```

Type LIST ENTER  
to list your program and see if it agrees.

Then type RUN ENTER  
and observe the output when you enter B, S, F = 2, 1, 9

It should be

X	SQUARE	CUBE	4-TH POWER
2	4	8	16
3	9	27	81
.	.	.	.
.	.	.	.
7	49	343	2481
8	64	512	4096
9	81	729	6561

If your headings do not seem to be lining up properly, change the spacing inside the quotation marks in the instruction

```
108 PRINT "X          SQUARE          CUBE          4-TH POWER"
```

until it pleases you.

Note that the instruction

```
150 IF X <= F THEN 120
```

makes a decision by comparing the values of X and F. (IF X <= F THEN 120) The IF instruction produces a branching in the path of the instructions, and takes the branch path (loop) instead of continuing on if the given condition is satisfied. This is indeed a powerful idea.

Try it again with B, S, F = 2, .5, 7.

As an experiment to test your understanding, let us insert another instruction.

125  $Y = V*Y$  (Note: The 108 **ENTER** deletes instruction 108, which  
 108 **ENTER** would have printed erroneous headings as previously  
 entered.)

You should forecast the output of the new program and insert new headings in instruction 108 before running the program.

```

100 PRINT "TYPE BEGINNING VALUE, STEP SIZE, FINAL VALUE";
105 INPUT B,S,F
106 CLS
108      (insert new headings here.)
110 X = B
120 Y = X*X
125 Y = Y*Y
130 PRINT X,Y,X*Y,Y*Y
140 X = X + S
150 IF X<= F THEN 120
160 PRINT
170 PRINT"  END OF TABLE"

```

The actual output for B,S,F = 2, .5, 10 is:

2	16	32	256
2.5	39.0625	97.6563	1525.88
3	81	243	6561
3.5	150.062	525.219	22518.7
4	256	1024	65536
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

As you experiment a bit, you will find the computer occasionally switching to an odd-looking number format, perhaps

1.23456E+08

This is called the "exponential" or "scientific" notation. The E+08 means to multiply 1.23456 by  $10^8$  (i.e., move the decimal point 8 places to the right).

Similarly 1.23456E-04 would mean to multiply by  $10^{-4}$  (i.e., move decimal point 4 places to the left).

4.21653E+12 = 4216530000000 (with a possible error of  $\pm 5$  in the 7th place)

7.65432E-08 = .0000000765432 (with a possible error of  $\pm 5$  in the 14th place)

If you are familiar with trigonometry and have Level II BASIC, you may wish to substitute.

```
108 PRINT "X", "SIN(X)", "COS(X)", "EXP(X)"
```

```
130 PRINT X, SIN(X), COS(X), EXP(X)
```

There is no need to retype the entire program; just get

```
READY
```

```
>_
```

and type in

```
108 PRINT "X", "SIN(X)", "COS(X)", "EXP(X)" ENTER
```

```
130 PRINT X, SIN(X), COS(X), EXP(X) ENTER
```

Run the revised program.

Note: we obtained the spacing desired in statement 108 by enclosing each column heading in quotation marks and separating the columns with commas, just as we did in statement 130 which prints the values. This is easier than using just one pair of quotation marks.








SUMMARY OF LESSON 2

What have you learned in Lesson 2?

Our new program extended your knowledge of the PRINT instruction to include the possibility of printing several different values from the same instruction, and of doing computation inside of the PRINT instruction itself.

You really do have a lot of computing power at your fingertips now.

KEYBOARD COMMANDS

<b>ENTER</b>	Enter line from screen to computer.
<b>CLEAR</b>	Clear the screen
<b>BREAK</b>	TRS-80, stop what you are doing and pay attention. Type CONT <b>ENTER</b> to continue.
	Backspace and erase.
<b>SHIFT</b> 	Erase entire line.
	(for Level I) } (Release the  key, to continue.)
<b>SHIFT</b> 	(for Level II) } STOP! (Depress any key, without using  to continue.)
NEW <b>ENTER</b>	Erase current program and data, clear screen and pay attention.
RUN <b>ENTER</b>	Run the program now in memory.
LIST <b>ENTER</b>	List the program now in memory.
SHIFT 	To get double-sized output ; <b>CLEAR</b> to return to single size.

BASIC INSTRUCTIONS DISCUSSED THUS FAR

X = B	} Computes value on right and stores it in the location named at left of equals sign.
Y = A*X*X + 4.6*X - 3.172	

PRINT X,Y,Z; T,X*Y	} Prints values of variables or expressions indicated. Also prints any message given inside of quata- tion marks. Semicolon is for narrow spacing, comma for wide spacing.
PRINT N; "VALUES OF X TOTAL"; Y	



INPUT "TYPE X VALUE"; X

*Types message in quotes, followed by a question mark and then accepts values from keyboard and stores them in location indicated.*

IF X <= F THEN 120

*Compares values of variables X and F. If X value is less than or equal to F value, then instruction 120 is executed next. Otherwise, the instruction following the IF instruction is executed next.*

+    *add*  
-    *subtract*  
\*    *multiply*  
/    *divide*

PRACTICE SESSION 2

As in Lesson 1, you should begin by typing in, listing, checking and running the main program under discussion in this Lesson, namely:

```

100 PRINT "TYPE BEGINNING VALUE, STEP SIZE, FINAL VALUE";
105 INPUT B,S,F
106 CLS
108 PRINT " X           SQUARE           CUBE           4-TH POWER"
110 X = B
120 Y = X*X
130 PRINT X,Y, X*Y, Y*Y
140 X = X + S
150 IF X <= F THEN 120
160 PRINT
170 PRINT "      END OF TABLE"

```

- 1a. Run the textbook program given above.
- 1b. Change instruction 140  $X=X+S$   
to 140  $X=X*S$   
and run the program. (Do you need to also change instruction 108?)
2. What happens if the program in the text is given B,S,F as 0, 2, 7 so that the step misses  $F = 7$ ? Does it stop at 6 or at 8?
3. a. Write a program of your own to create a table of values of
 
$$X \quad X+3 \quad (X+3)^2$$
 for  $X = 1, 2, 3, \dots, 10$
- b. Modify your program so it will accept input values B,S,F and print out
 
$$X, X+3, (X+3)^2$$
 for X values from B to F in steps of S.
4. Write some programs of your own choice using the instructions we have discussed.
5. Write a program to INPUT the radius  $R$ , of a sphere, and print out on one line the radius, volume and surface area of the sphere using

$$\text{Volume} = \frac{4}{3} \pi R^3$$

$$\text{Surface Area} = 4\pi R^2$$

(Where  $\pi = 3.14159\dots$ )

6. Modify your program of problem 5 to INPUT values: B, S, F (for Beginning value, Step size and Final value) and PRINT a table of values of the radius, volume and surface area of spheres of radius B, B+S, B+2S, B+3S,...,F.
7. Write a program to convert Fahrenheit temperature to Celsius.

$$C = \frac{5}{9} (F - 32)$$

8. Write a program to convert temperature in degrees Celsius to Fahrenheit temperature.

$$F = \frac{9}{5} (C) + 32$$

9. Rewrite instruction 108 in the program that produces powers of numbers, so that the headings are valid after instruction 125 Y = Y\*Y is also inserted. Check it by using B, S, F = 2, 1, 4
10. If you have LEVEL II BASIC, try adding the following statement to the program that makes tables of x, x<sup>2</sup>, x<sup>3</sup>, x<sup>4</sup>

5 DEFDBL X, Y  
and RUN the program with B, S, F = 2, .5, 6

Note that we no longer have problems with  $4.5^4 = 410.0625$  being rounded to 410.063 since we are no longer restricted to 6 decimal digits in our arithmetic. The DEFDBL X,Y instruction permits the TRS-80 to compute with and display up to sixteen decimal digits of accuracy when needed. We shall discuss this further when the additional accuracy is really needed. (What happens to the four columns of output?)

11. Write a program to create tables of values of  
x, x<sup>3</sup>, x<sup>6</sup>, x<sup>9</sup>  
with proper headings for various values of B, S, F typed in by the user.
12. What output occurs in the program to produce a table of x, x<sup>2</sup>, x<sup>3</sup>, x<sup>4</sup> if some or all of B, S, F are negative? Try B, S, F = 2, -.5, -3 and B, S, F = -2, 1, 5 and B, S, F = -1, -1, -6.
13. What happens when some of B, S, F are negative in the table of powers program that includes 125 Y = Y\*Y ?

- 14a. If your BASIC includes a square root function `SQR( )` try using  
`120 Y = SQR(X)`  
`130 PRINT X,Y,SQR(Y),SQR(SQR(Y))`  
 in the text program.
- b. If you don't have a `SQR( )` function, try  
`120 Y = X*X*X`  
`130 PRINT X,Y,X*Y,Y*Y`  
 Be sure to alter instruction 108 to print a correct heading for the current table columns in either (a) or (b).
15. What would happen to the program if the instruction that prints the heading were changed to  
`123 PRINT "X            SQUARE            CUBE            4-TH POWER"`  
 Try it out and see. After you understand what happened, and why, delete instruction 123 by typing

`123` ENTER

Notice: If mathematics frightens you, you can laugh at your fears now. With a computer to do the arithmetic, all you need to do is type in the formula and the computer will do the arithmetic for you. It really is easy to do problems that used to stump you. Try it; you'll like it!

16. If you like mathematics, you may wish to start by proving that the area of a rhombus (a "stepped-on" square) is given by  $\frac{1}{2}CD$  where C and D are the lengths of its diagonals. If you don't like mathematics, just believe us that the area of a rhombus is half the product of its diagonals. (OK?)

Write a program that will accept input values of C and D and print out the message

THE AREA OF A RHOMBUS HAVING DIAGONALS \_\_\_\_ AND \_\_\_\_ IS \_\_\_\_\_.  
 with the blanks filled in with the proper values. After your program is running properly, put in some extra instructions that will print out the message

THE VALUES OF C AND D MUST BOTH BE POSITIVE.

if anyone uses negative or zero values as input.

Hint:

```
100 INPUT "PLEASE TYPE VALUES OF DIAGONALS C,D="; C,D
110 IF C < 0 GOTO 500
120 IF D < 0 GOTO 500
130 A = .5*C*D

150 PRINT "THE AREA OF A RHOMBUS HAVING DIAGONALS ";C;"AND";
    D;"IS";A
160 PRINT
170 GOTO 100
```

(continued on next page)

```

500 PRINT "THE VALUES OF C AND D MUST BOTH BE POSITIVE."
510 PRINT
520 PRINT
530 GOTO 100

```

17. If  $N$  is the number of sides in a regular polygon and  $L$  is the length of each side, then the radius of the inscribed circle is

$$\frac{L}{2} \frac{\cos K}{\sin K} \quad \text{where } K = \frac{\pi}{N}$$

The radius of the circumscribed circle is

$$\frac{L}{2 \sin K} \quad \text{where } K = \frac{\pi}{N}$$

The area of any circle is given by  $\pi r^2$  where  $r$  is its radius and  $\pi = 3.14159\dots$  You should be able to create a program that would list

Number of Sides / Area of Inscribed Circle / Area of Circumscribed Circle

for polygons having 3 or more sides each of length 1 unit. Do so. Examine the output and explain what is happening. If you don't know any trigonometry, do not let it disturb you. You can still write this program. Use

```

K = 3.14159/N
I = .5*COS(K)/SIN(K)
C = .5/SIN(K)

```

where  $I$  stands for the area of the inscribed circle and  $C$  stands for the area of the circumscribed circle.

18. A tetrahedron (triangular pyramid) has four faces each of which is an equilateral triangle whose sides are each of length  $L$ .

$$\text{The surface area} = 1.73205L^2$$

$$\text{The volume} = 0.11785L^3$$

At  $L = 1$ , the surface area is larger than the volume.

At  $L = 20$ , the surface area is smaller than the volume.

Find the value of  $L$  that makes the number of square units in the surface as nearly as possible equal to the number of cubic units in the volume.

19. For what radius does a sphere have the same surface area (in square units) as volume (in cubic units)?

$$\text{Vol} = \frac{4\pi}{3} R^3$$

$$\text{Surface} = 4\pi R^2$$

20. If  $(X1, Y1)$  and  $(X2, Y2)$  are the coordinates of two points, the distance between the two points is given by the formula

$$D = \sqrt{(X1-X2)^2 + (Y1-Y2)^2}$$

Write a program to accept the coordinates of two points and print out the distance between the points. Does the formula work if some of the values are negative?



22. CHALLENGE PROBLEM: A job pays you \$1 for the first 8-hour day's work, \$2 for the second day's work, \$4 for the next day's work, \$8 for the next day's work, etc., doubling your day's wages each day, if you continue to work without being tardy or absent. It starts over at \$1 any day you are late or missing.

Find how many days you will have to work, without a tardy or absence to earn a total of a million dollars. How much would you earn the next day, if you were still on time?

You may write your own program if you prefer, or you may use the program below:

```

10 PRINT "DAY#", "PAY ON THAT DAY", "TOTAL PAY THUS FAR"
100 S = 0
110 D = 1
120 P = 1
130 S = S+P
140 PRINT D,P,S
150 D = D+1
160 P = P+P
170 GOTO 130

```

After you run your program, modify it so that you can INPUT a value of T for the number of minutes tardy in step 170 and then send the program to 130 if T = 0 but to 120 otherwise. Use T = 480 if absent.

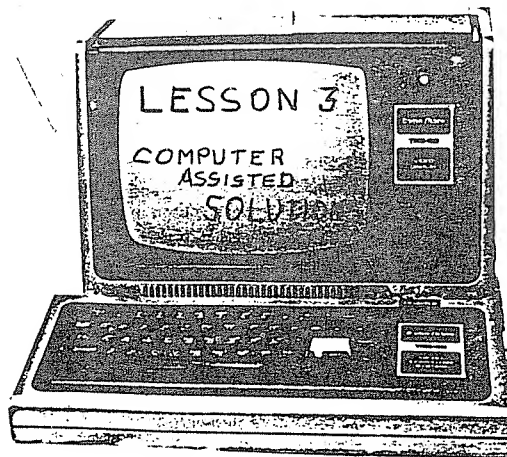
```

170 INPUT T
180 IF T = 0 THEN 130 ELSE 120

```

What effect does this have?

23. CHALLENGE PROBLEM: It is easy to observe that the sum of the first four natural numbers is  $1+2+3+4 = 10$  or that the sum of the first seven integers is  $1+2+3+4+5+6+7 = 28$ . How many successive natural numbers beginning with 1 would you have to add together to reach or just exceed 1000?
24. CHALLENGE PROBLEM: Bob drops a superball from a 4th floor window 42 feet above the parking lot below. If on each bounce the ball ascends to half the height it fell on that bounce, how far will the ball travel (both up and down) when it hits the ground for the 10th time? The 50th time? The 100th time?



All right—it is now time to learn to do something you can't find in a set of tables and can't do without a computer—at least not conveniently.

Now, let's introduce a new instruction by reprogramming a "Table Maker" of Lesson 2.

A very useful instruction is the FOR...NEXT instruction. Try the following program on your computer.

```
100 FOR K = 3 TO 10
110   S = K*K
120   PRINT K,S,K*S,S*S
130 NEXT K
140 PRINT "END OF TABLE"
```

The FOR...NEXT instruction sets  $K = 3$  and executes whatever instructions lie between FOR and NEXT; then  $K$  is advanced to  $K = 4$  and the instructions between FOR and NEXT are again executed; then similarly for  $K = 5, 6, 7, 8, 9, 10$ . When  $K = 10$ , the instructions between FOR and NEXT are again executed with  $K = 10$ . After reaching 130 NEXT  $K$  with  $K = 10$ , the program continues to instruction 140.

After you have run the above program, modify it to

```
80 PRINT "TYPE BEGINNING VALUE, END VALUE";
90 INPUT B,E
100 FOR K = B TO E
110   S = K*K
120   PRINT K,S,K*S,S*S
130 NEXT K
140 PRINT "END OF TABLE"
150 PRINT
160 GOTO 80
```

RUN the program for several different values of  $B$  and  $E$ . Now we are ready to write a more sophisticated program.

Let's write a program to help us solve messy equations.

Notice that no one said the program would solve equations. It won't. But it will do all the hard work and messy arithmetic for you. All you'll need to do is to tell it where to hunt for the roots, and it will do the hunting. Does that sound fair? Let's try it. If mathematics bores you, just hang in there for now. We'll be doing computer assisted art in Lesson 4 but for now I want you to realize how easy it really is to do mathematics with the help of a computer. If you can do 6th grade arithmetic, you can do this. Keep on reading, please.

It may not have occurred to you that computer programs have to be "built." They do not spring full grown from the mind, but develop slowly--and are modified as they develop--until a final program you would be willing to share with a colleague eventually results. One of the primary rules for good programs is

```

.....
Make your program run first,
.....
then make it fancy.
.....

```

This is good advice. Let's use it.

Our problem is to write a program to help solve the equation

$$5X^3 + 3X^2 - 2X - 5 = 0.$$

We shall first write a program that accepts a value of X as input, and displays the value of X along with the corresponding values of

$$Y = 5X^3 + 3X^2 - 2X - 5.$$

To solve the equation, we must make  $Y = 0$  (or as close as we can get).

The following preliminary program seems to provide the required evaluation.

```

210 PRINT "TYPE X-VALUE PLEASE";  ENTER
215 INPUT X  ENTER
240 Y = 5*X*X*X + 3*X*X - 2*X - 5  ENTER
250 PRINT X,Y  ENTER
280 PRINT  ENTER
290 GOTO 210  ENTER

```

Don't forget to type LIST  ENTER to check your program.

Then type RUN  ENTER

The program will print

TYPE X-VALUE PLEASE ?

If you type  0  ENTER

It will respond

-5

TYPE X-VALUE PLEASE?

If you type 1 

It will respond

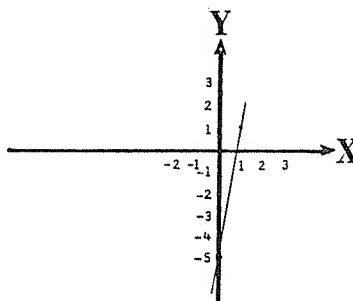
1 1

TYPE X-VALUE PLEASE?

The change of sign in Y from -5 to +1 shows, since Y is a continuous function, that there is a root somewhere between  $X = 0$  and  $X = 1$  — possibly nearer  $X = 1$ .

If we were to graph  $Y = 5x^3 + 3x^2 - 2x - 5$  we would know that for

X	0	1
Y	-5	1



Our observation suggests that for some X-value between  $X = 0$  and  $X = 1$ , the value of Y must be zero. It is this X-value that corresponds to  $Y = 0$  that we seek.

Let's try 0.8 

The computer responds

.8 -2.12

TYPE X-VALUE PLEASE?

So we now know there is a root between  $X = 0.8$  and  $X = 1$  (Why?)

so we type 0.9 

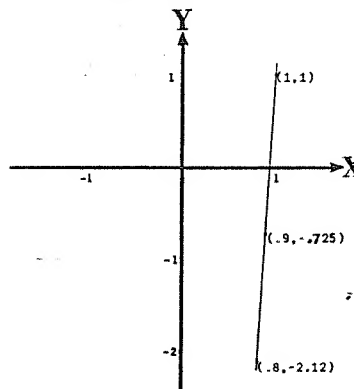
and obtain

.9 -.725

so the sign change (and root) comes between  $X = .9$  and  $X = 1$

We can now enter .95 

and if the corresponding Y is positive, follow this with .94 etc'. Between .944 and .945 we eventually obtain the root accurate to 5 or 6 significant digits. Try a few values yourself.



We can devise an even better helper program to solve equations. Let's modify our program so it accepts two values B and S (for Beginning value and Step size) and then prints out 11 pairs of values of X and Y before looping back to get another pair of starting values.

If you aren't sure why this will be a convenient change, just wait until you try the program given below.

We shall use our new FOR....NEXT instruction. It is really a pair of instructions

```
FOR K=1 TO 11
  . . .
  . . .
NEXT K
```

The instruction FOR K=1 TO 11 sets K=1 and performs the instructions between FOR and NEXT (which may or may not involve K).

The NEXT K instruction then adds 1 to K (making K=2) and again performs the instructions between FOR and NEXT. Then 1 is again added to K, producing K=3, and the instructions between FOR and NEXT are executed again, and again, and again until finally K=11. The instructions between FOR and NEXT are performed once more, but this time whatever instruction directly follows the NEXT is executed.

The wee program

```
100 FOR K=1 TO 20
110   PRINT K,K*K
120 NEXT K
```

will produce a table of integral values of K and  $K^2$  for K between 1 and 20 inclusive.

The following program produces rather different results in Level I and Level II BASIC.

Try it on your computer.

```
100 FOR K=0 TO 20 STEP .5
110   PRINT K,K*K
120 NEXT K
```

On Level I BASIC, it produces a table whose entries are 0's. This is because in Level I BASIC, STEP size must be an integer and STEP .5 is truncated to STEP 0.

Level II produces a table of  $K, K^2$  for K between 0 and 20 with values changing by .5.

Try it if you are unsure. The best way to learn about your computer is to compute.

Let us return to our first version of

```

210 PRINT "TYPE X-VALUE PLEASE";
215 INPUT X
240 Y = 5*X*X*X + 3*X*X - 2*X - 5
250 PRINT X,Y
280 PRINT
290 GOTO 210

```

Let us combine some of the tricks we learned in Lesson 2 to produce a table of eleven values of X and  $Y = 5*X*X*X + 3*X*X - 2*X - 5$  starting with X = B and increasing the X value by S on each line.

Second try:

```

210 PRINT "PLEASE TYPE BEGINNING VALUE, STEP SIZE";
215 INPUT B,S
220 X=B
230 FOR K=1 TO 11
240   Y = 5*X*X*X + 3*X*X - 2*X - 5
250   PRINT X,Y
260   X=X+S
270 NEXT K
280 PRINT
290 GOTO 210

```

LIST  and check your program.

Then type RUN

The program will print

PLEASE TYPE BEGINNING VALUE, STEP SIZE?

and you type                      0, 1

The output will be:	0	-5	← (Note sign change!)
	1	1	
	2	43	
	3	151	
	4	355	
	5	685	
	6	1171	
	7	1843	
	8	2731	
	9	3865	
	10	5275	



Since there is a sign change between  $X = 0$  and  $X = 1$ , you next type 0, .1 ENTER which produces

0	-5
.1	-5.165
.2	-5.24
.3	-5.195
.4	-5
.5	-4.625
.6	-4.04
.7	-3.215
.8	-2.12
.9	-.724999
1	1 ←---- (Note sign change!)

A sign change between .9 and 1.0 indicates a root there, so you type .9, .01 ENTER.

Then later you type .94, .001 ENTER which shows a root between .944 and .945.

.94	- .0762806
.941	- .0593705
.942	- .0424247
.943	- .0254455
.944	-8.43143E-03
.945	8.61549E-03 ←---- (Note sign change!)
.946	.0256987
.947	.0428152
.948	.0599666
.949	.0771518
.95	.0943718

So now we know that the equation  $5x^3 + 3x^2 - 2x - 5 = 0$  has a root between  $x = .944$  and  $x = .945$ . You can easily obtain 5 or 6 places of accuracy, if needed, but 3-place accuracy is frequently enough.

To solve  $7x^4 + 4x^3 + 2x^2 - 9x - 100$ , you need only to change line 240 to

240 Y = 7\*X\*X\*X\*X + 4\*X\*X\*X + 2\*X\*X - 9\*X - 100

Starting values 0, 1 (and also 0, -1) will each be helpful in solving this problem.

Computer experts will prefer to write line 240 as

$$240 \quad Y = (((7*X + 4)*X + 2)*X - 9)*X - 100$$

which saves both typing and, more important, computing time, since it uses only four multiplications instead of 10, each time Y is evaluated. If you have had a course in algebra, please multiply out the expression to see that the same results are obtained. You can solve many different equations by changing line 240.

What about solving  $x^3 - 4.9x^2 + 6.6x - 2.6 = 0$  by changing line 240 to

$$240 \quad Y = ((X - 4.9)*X + 6.6)*X - 2.6$$

Note that input B, S = 0, 1 produces three changes in sign. Find all three roots accurate to 4 or 5 significant digits.

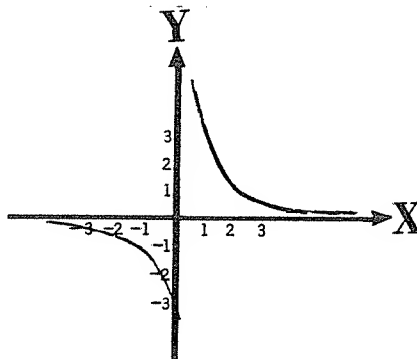
If you have Level II BASIC, you may wish to try  $Y = e^x \cos x^2 - 2.6$

$$240 \quad Y = \text{EXP}(X)*\text{COS}(X*X) - 2.6$$

This program is really quite a valuable tool. It will help you to solve almost any continuous (e.g., *no factor in the denominator that might be zero for X inside the interval in which you are seeking a root*) function set equal to zero.

Polynomials come under this classification—even polynomials in  $\sin(x)$ ,  $\cos(x)$ ,  $e^x$ , as well as polynomials in X. So do many other functions, but not equations like

$$\frac{3}{2x-1} = 0 \quad \text{since } y = \frac{3}{2x-1} \text{ is discontinuous at } x = 1/2$$



Remember: to change the equation being solved, you only need to change instruction 240 to

$$240 \quad Y = (\text{whatever continuous function you are trying to make zero}).$$

CORRECTING ERRORS discovered at RUN time.

You may have noticed that the TRS-80 will frequently detect errors in spelling or punctuation of instructions and will warn you of these errors by printing an ERROR message; consider the following erroneous program:

```
100 FOT K=1 TO 10
110 PRINT K,K*K
120 NEXT K
```

in which FOR is misspelled FOT in line 100. If you type

```
RUN ENTER
```

the computer responds

```
?SN ERROR IN 100
READY
100 _
```

The first line tells you that there is an ERROR in statement 100. We shall talk more about ERROR types later on - for now you can usually spot the error by looking at statement 100.

For certain (but not all) errors, the computer will shift automatically into EDIT mode. This is indicated by displaying the line number where the ERROR has been detected just below the ERROR IN 100 message.

If you depress the [L] key (without ENTER), the entire line will be displayed, and the line number repeated

```
100 FOT K=1 TO 10
100 _
```

If you tap the [space bar] the line will appear character by character until the underline marks where the ERROR is:

```
100 FOT K=1 TO 10
100 FO_
```

Now depress [C] (which stands for change); nothing visible will happen, but if you now depress the desired character [R] it will appear. If there are other errors, correct them too. Then depress ENTER

Type LIST ENTER and you will find the correction was made.

If you should need to delete a character depress [D] instead of [C], the deleted character will appear between ! ! symbols. To insert a character depress [I] instead of [C] and then depress the desired

character. If you wish to change, delete or insert several characters depress the number key before depressing the **C** , **D** , **I** key. Thus **3 I** will enable you to insert 3 characters. This is only part of a set of powerful EDIT instructions available in Level II BASIC. We shall discuss them in Lesson 8 , but you may peek now if you wish.

In many cases it is much simpler to merely retype the line, rather than using the EDIT instructions. In such a case if you type

```
RUN ENTER and get
```

```
?SN ERROR IN 100
READY
100 _
```

Simply depress **ENTER**

This will display line 100, and display the >\_ which you must have to enter an instruction. Depressing **ENTER** will display

```
?SN ERROR IN 100
READY
100 FOT K=1 TO 10
>_
```

Then you merely retype the instruction

```
>100 FOR K=1 TO 10 ENTER
```

The critical things to remember are:

The computer indicates it is ready to accept data by displaying ?  
Do not type anything but data if you have a ? displayed.

The computer indicates it is ready to accept an instruction by displaying >\_  
Do not type an instruction unless you have a >\_ displayed.

If in doubt depress **BREAK** followed by **ENTER** to get

```
READY
>_
```

## SUMMARY OF LESSON 3

We learned about the

```
FOR K = 3 TO 10
```

```
.
```

```
.
```

```
.
```

```
NEXT K
```

and

```
FOR K = 0 TO 20 STEP .5
```

```
.
```

```
.
```

```
.
```

```
NEXT K
```

instructions, and used them to develop a powerful program that can be used to help solve continuous equations.

We also learned how to correct errors that are detected by the computer at RUN time.

Perhaps the most important thing we learned in Lesson 3 is that mathematical problems are not inherently difficult. Even if mathematics troubled you in the past, with the able assistance of a microcomputer YOU CAN use plain old common sense to solve problems that would cause a college mathematics major difficulty if he tried to solve them without a computer.

PRACTICE SET 3

In problems 1 to 10, use the second equation solver program to help you find solutions of the problems given below.

1.  $x^5 + 3x^4 + 2x^3 + 5x^2 + 3x - 25 = 0$
2.  $x^5 - 31 = 0$
3.  $7x^4 + 6x^3 + 10x^2 - 5x - 1 = 0$
4.  $4x^3 - 8x^2 - 29x - 13 = 0$  (Find three roots between -10 and +10.)
5.  $x^4 - 3x^2 - 4 = 0$
6. a.  $x^4 - x^3 - 5x^2 - x - 6 = 0$   
b.  $x^4 - x^3 - 5x^2 - x - 5 = 0$
7.  $2x^4 + x^3 - 8x^2 - x + 6 = 0$
8.  $6x^4 + 5x^3 - 14x^2 + x + 2 = 0$
9.  $3x^3 + 4x^2 - 12x - 16 = 0$
10.  $4x^4 - 13x^2 + 3 = 0$
11. a.  $2x^4 - x^3 - 15x^2 + 6x + 17 = 0$   
b.  $2x^5 - x^4 - 6x^3 + 3x^2 + 4x - 2 = 0$
12. Change the constant terms in some of the above equations and try again.
13. If you have a Level II BASIC, try some trigonometric equations like

$$X \cdot \sin(X) - 2 = 0$$

$$X \cdot \exp(X) \cdot \cos(X) + X \cdot X - 4.5 = 0$$

The fascinating thing is you don't really need to understand trigonometry to solve the given (rather carefully selected) equations.

14. Forecast the output of the following program, then RUN it to check your forecast.

```
100 FOR X=1 TO 10 STEP .2
110 PRINT X,X*X, X*X*X
120 NEXT X
```

Be sure to think about whether your computer has Level I BASIC or Level II BASIC.



15. Use the program

```

90 INPUT "PLEASE TYPE BEGINNING VALUE, END VALUE, STEP SIZE=";B,E,S
100 FOR X = B TO E STEP S
105   Y=X*X
110   PRINT X,Y,X*Y,Y*Y
120 NEXT X
130 PRINT
140 GOTO 90

```

16. The following program is supposed to accept a positive integer value  $N$  and print out the values of  $N$  and  $N!$  -  $1*2*3*...*N$ . It seems to work the first time through, but then prints out non-sense values for  $N!$ . Try it and see. Then debug and repair the program.

```

100 PRINT "N", "N-FACTORIAL"
110 PRINT
120 F=1
150 INPUT "TYPE INTEGER WHOSE FACTORIAL YOU WISH" ; N
155 IF N<0 THEN 150
160 IF N=0 PRINT N, F
170 IF N<INT(N) THEN 150
200 FOR K = 1 TO N
210   F = F*K
220 NEXT K
250 PRINT N, F
260 GOTO 150

```

*These test the input data...*  
NOTE: INT(N) produces the integer portion of N, so  
170 IF N < INT(N) THEN 150 looks at N and if N is not an integer, the program goes back to 150. Similarly, if N is negative, instruction 155 IF N < 0 THEN 150 sends the program back to instruction 150.

17. The following program is supposed to accept two integers N, D and determine whether or not D is a factor of N. Does it work? If so, why? If not, fix it. The heart of the program is lines 130, 140.

```

100 INPUT "TYPE IN TWO POSITIVE INTEGERS"; N, D
110 IF N<D THEN 500
120 IF N=D THEN 600
130 Q=INT(N/D)
140 IF D*Q <> N THEN 400
200 REM PROGRAM COMES HERE ONLY IF D IS A FACTOR OF N
210 PRINT D; "IS A FACTOR OF";N
220 PRINT
230 GOTO 100
400 REM PROGRAM COMES HERE ONLY IF D IS NOT A FACTOR OF N
410 PRINT D; "IS NOT A FACTOR OF"; N
420 PRINT
430 GOTO 100

```

(continued on next page)

17. (continued)

```

500 PRINT N;"=N IS SMALLER THAN D=";D;"HENCE D CAN'T BE A FACTOR OF N"
510 X=N
520 N=D
530 D=X
540 GOTO 130

600 PRINT "THE VALUES BOTH EQUAL"; N, "HENCE EACH DIVIDES THE OTHER"
610 PRINT
620 GOTO 100

```

18. In Lesson 2 you wrote programs to change temperature in Fahrenheit to temperature in Celsius and vice versa. Combine the ideas of your two programs into a single program that accepts B, S, E as Beginning value, Step size, and End value for T and then writes out two sets of tables side by side (four columns) using  
 PRINT T;C,T;F      *←————(Note use of ; , ; in output here.)*  
 where T runs from B to E in steps of S and where  
 C = (the Celsius temperature corresponding to T degrees in Fahrenheit)  
 F = (the Fahrenheit temperature corresponding to T degrees Celsius)
19. Write a program to INPUT the hourly rate, R, and the number of hours, H, worked in a given week and print out the employee's gross wages for that week, assuming (s)he is paid time and a half for all time over forty hours in a given week.  
 The output statement might be:  
 PRINT "\$";W;"FOR";H;"HOURS AT BASIC HOURLY RATE OF";R  
 but you'll have to scratch your head a bit to compute the total for W, including the overtime payments, if any. Don't make the error of subtracting ten hours of "undertime" for an employee who only works 30 hours. It does not work that way.
20. Write  $X^4 - 7X^3 + 5X^2 + 3X + 71$  as a series of nested parentheses.
21. Express  $((3 * X - 7) * X + 5) * X + 2$  in a form without parentheses.
22. CHALLENGE PROBLEM: The sequence 1,1,2,3,5,8,13,21,... is created by writing 1,1 and then each succeeding term is the sum of the preceding two terms, 1+1=2, 1+2=3, 2+3=5, etc.  
 Write a program to compute the first 100 or so terms of this "Fibonacci sequence."
23. CHALLENGE PROBLEM: Solve  $X^2 - \cos X = 0$
24. CHALLENGE PROBLEM:  $4x^8 - 2x^7 + x^6 - 3x^4 + x^2 - x + 1 = 0$   
 presents an interesting puzzle for the mathematically-oriented reader.

25. CHALLENGE PROBLEM: A dog is chasing a rabbit. The rabbit takes three jumps in the same length of time the dog takes two jumps, but each rabbit jump covers only half of the distance of a dog jump. The rabbit was 13 rabbit jumps ahead of the dog when the dog first spotted the rabbit and started after it. If both go in a straight line, how many more jumps will the rabbit take before the dog catches it?
26. CHALLENGE PROBLEM: How much must you invest now in an account paying 9% interest compounded quarterly to have \$100,000 when you are 70 years old? (If you don't want to give your age away, figure it out for S. Marguerite who is 22 years old.)
27. CHALLENGE PROBLEM: Legend states that in 1626 Manhattan Island was purchased from the Indians for \$24. If that \$24 had been invested at 12% interest (a modest interest rate at that time) what would the \$24 be worth today?  
(How does this compare with the value of the land in Manhattan Island today?)
28. Here are two small "research" problems for you.  
CHALLENGE PROBLEM: The four-digit number 9801 has the unusual property that if you take the two-digit number formed by the first two digits, 98, and add it to the number represented by the last two digits, 01, then  $98 + 01 = 99$  and  $(99)^2 = 9801$ . Find all four-digit numbers that have this property.
- 29.a) CHALLENGE PROBLEM: A 4 by 4 rectangle (square) has the unusual property that its perimeter (distance around it) is the same as its area (both are 16). Your problem is to find another rectangle that also has the property that its length and width are both integers and its perimeter ( $2*(L+W)$ ) is equal to its area ( $L*W$ ).
- b) CHALLENGE PROBLEM: If you like mathematical thinking, see if you can prove that the two rectangles you found above are the only two such rectangles that exist.
30. CHALLENGE PROBLEM: The song "Twelve Days of Christmas" mentions various gifts that "my true love gave to me." Let's interpret the song so that "on the third day of Christmas, my true love gave to me three French hens, two turtle doves and a partridge in a pear tree." My true love gave me six presents (3 French hens, 2 turtle doves, and a partridge, if we don't count the pear tree as a gift). How many gifts in total did my true love give to me during the twelve days of Christmas? Write a program to determine the sum.  
  
Answer: 364, if you do not count the pear tree as a gift.

# LESSON 4

## ART & GRAPHICS

Computers are fun. They make great adult toys. As my daughter says, "The main difference between men and boys is the price of their toys." Well, a modern computer costs less than many adult toys — a Jaguar XKE, a sailboat, a motor boat, a golf cart or even a set of clubs can cost more than a computer. Of course, the computer is a very useful device (but so is an XKE) so let's just look at how to have fun with it.

If you are curious, you may have doped out how the little surprise program of Problem 8, Lesson 1, worked. Let's look at it.

(Remember to depress **ENTER** at the end of each line.)

### Level II BASIC

```
NEW ENTER
10 PRINT @ RND(1000), "HI SUZANNE"
15 IF RND(0) < .05 THEN CLS
20 GOTO 10
RUN
```

### Level I BASIC

```
NEW ENTER
10 PRINT AT RND(1000), "HI SUZANNE"
15 IF RND(0) < .05 THEN CLS
20 GOTO 10
RUN
```

Line 10 contains two new ideas. One of these is the idea of a random number generator. TRS-80 contains two types of random number generators: One is called RND(0). RND(0) produces a random decimal number between 0 and 1.

The other is RND(n), where *n* is a positive integer. RND(n) produces a random integer between 1 and *n* inclusive.

To see how they work, use the programs on the following page.

```

1000 PRINT RND(0);
1010 GOTO 1000
RUN 1000 ENTER +

```

After you have a screen full of that, try

```

1000 PRINT RND(100);
1010 GOTO 1000
RUN 1000 ENTER

```

Later, change instruction 1000 to

```

1000 PRINT RND(10);

```

(Note: this still uses instruction 1010 GOTO 1000.)

and type RUN 1000 ENTER again.

Each time you call RND(*n*) a random number is produced that is different from the random number produced last time, even though the same *n* is in the parentheses.

The second new idea in the instruction

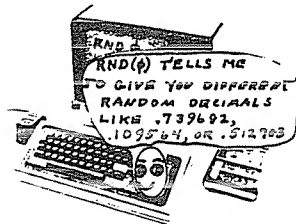
```

10 PRINT @ RND(1000), "HI SUZANNE"

```

is the PRINT AT \_\_, "\_\_\_\_\_" instruction.

For purposes of the PRINT AT \_\_, "\_\_\_\_\_" the screen is divided into 1024 cells, sixteen rows of 64 elements each. Each cell exactly fits one letter plus the space between rows for that letter. The cells are numbered from 0 to 63 in the first row, 64 to 127 in the second row, etc. (See illustration on the following page.)



<sup>+</sup> This uses RUN 1000 ENTER in place of RUN ENTER to make the computer skip any program in 10, 15, 20,...,999, and start RUNNING at statement 1000. A neat trick.

HI SUZANNE

Try the program

```

1000 INPUT N
1005 CLS
1010 PRINT @ N, "*" MARKS LOCATION"; N;
1020 GOTO 1000
RUN 1000

```

(Use AT in place of @ on Level I BASIC.)

The instruction

PRINT  $\left(\begin{smallmatrix} \text{AT} \\ @ \end{smallmatrix}\right) n$       use  $\begin{cases} \text{AT in Level I BASIC.} \\ @ \text{ in Level II BASIC.} \end{cases}$

will start printing at location  $n$  and print whatever is called for. It prints any symbols it finds inside of quotes as symbols and the values of any variables it finds outside of quotes, providing the proper commas and/or semicolons are included.

For example, instruction 10 of the program in Problem 8 of Lesson 1

```
10 PRINT  $\left(\begin{smallmatrix} @ \\ \text{AT} \end{smallmatrix}\right)$  RND(1000), "HI SUZANNE"
```

combines these two instructions. RND(1000) generates a random integer between 1 and 1000.

```
PRINT  $\left(\begin{smallmatrix} @ \\ \text{AT} \end{smallmatrix}\right)$  RND(1000), "HI SUZANNE"
```

then prints whatever is in the quotes, starting at whatever random location RND(1000) generated. The next time, a different random integer is generated. The result is the statement in quotes is flashed all over the screen.



If you wish a snow storm, try

```
NEW  ENTER
1000 PRINT @ RND(1020), "*";
1020 GOTO 1000
```

To keep the screen from scrolling use ; at the end of line 1000.

To occasionally wipe the screen clear, add

```
1010 IF RND(0)<.01 THEN CLS
```

The instruction `RND(0)` generates a random decimal number between 0 and 1. If the random number generated is less than .01 (which it will be about once in 100 numbers on the average), the "clear screen" instruction `CLS` is executed. Otherwise, `CLS` is ignored. In either case, the instruction `1020 GOTO 1000` is executed next.

Here are some interesting programs. Try them on your TRS-80.

```
NEW  ENTER
5 CLS
10 FOR L = 0 TO 960 STEP 64
15   FOR K = 1 TO 61
20     PRINT @ K + L, "HI your name"
25   NEXT K
30 NEXT L
40 GOTO 10
```

After you try that with several different names, change instruction 15 to

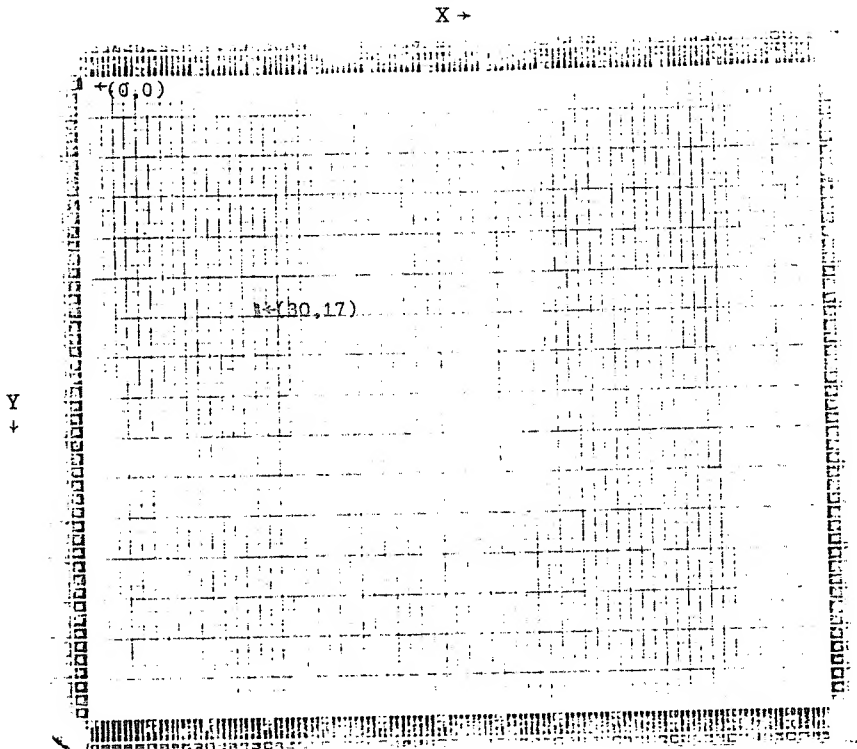
```
15 FOR K = 62 TO 5 STEP -1
```

and `RUN` it again.

Here is another FUN program. Please try it, then analyze what happens and why. The `CHR$( )` instruction, discussed later in this lesson, displays the character corresponding to the value in the parenthesis.

```
NEW  ENTER
1000 CLS
1100 N=1
1200 PRINT @ 970, " MOBILE ART FORM = "; N ;
1300 FOR K = 129 TO 191
1400   PRINT @ RND(959), CHR$(K);
1500 NEXT K
1600 N = N+1
1700 FOR Q = 0 TO RND(N) : NEXT Q
1800   IF N < 50 THEN 1200 ELSE 1000
```

Your TRS-80 has much finer graphic capabilities than we have been using in `PRINT (@) _____, "_____"`. Actually, each letter-sized cell is broken up into six smaller rectangular cells, or *pixels*, counting the space between lines, and each cell is individually addressable. The system of numbering from 0 to 1023 is too coarse for this use. The individual spots are addressed by giving an X, and Y-coordinate with  $0 \leq X \leq 127$  and  $0 \leq Y \leq 47$ . The (0,0) point is in the upper left edge of the screen. X increases to the right. Y increases downward.



So, (30, 17) is 30 spaces to the right and 17 spaces down from (0,0), as shown above.

Try

```
1000 INPUT "X,Y=" ; X,Y
1010 SET(X,Y)
1020 GOTO 1000
RUN
```

There are three instructions used with these spots:

SET(X,Y) This lights point (X,Y).

RESET (X,Y) This darkens point (X,Y).

POINT(X,Y) This is used to determine whether or not point(X,Y) is lighted. If lighted, it returns "logical yes", otherwise, "logical no", as used in an IF...THEN instruction.

```
1000 CLS
1100 FOR K = 0 TO 47
1200 SET(K,K) : SET(K+60,K)
1300 NEXT K
RUN
```

After you have RUN the above program, and understand why it produces two wiggly diagonal lines, try adding

```
125 SET(120-K,K) : SET(60-K,K)
```

Before you run the new program

```
1000 CLS
1100 FOR K = 0 TO 47
1200 SET(K,K) : SET(K+60,K)
125 SET(120-K,K) : SET(60-K,K)
1300 NEXT K
```

see if you can forecast the pattern that will be produced. Then (and only then) type RUN **ENTER** to check your forecast. This is one of the best ways to learn about computers.

IF POINT(X,Y) THEN RESET(X,Y) will test (X,Y) and turn it off, if it is on. However, RESET(X,Y) also turns off point (X,Y) and takes less time.

Later, you will find uses for POINT(X,Y) in programs in which you will need to test whether or not a given pixel is lighted.

The following program will help you become familiar with the location of the various (X,Y) rectangles (pixels). It lights a rectangle chosen at random, and then permits you to input your estimate of the coordinates (X,Y) of that rectangle. If you miss, a rectangle will blink at the coordinates you chose to show you its location with respect to the unblinking target. You may then input another guess.

```

1000 R = 10 + RND(30)
1100 C = 20 + RND(60)
1200 CLS
1300 SET(C,R)
1400 INPUT "PLEASE TYPE YOUR ESTIMATE OF (X,Y) AS  X,Y ="; X,Y
1500 IF(X=C) AND (Y=R) THEN 4000

2000 FOR K=1 TO 2000
2100     SET(X,Y)
2200     RESET(X,Y)
2300 NEXT K
2400 GOTO 1200

4000 REM HERE IF ESTIMATE IS CORRECT.
4100 PRINT @ 70, "YOU ARE CORRECT. CONGRATULATIONS."
4200 FOR Q=1 TO 800 : NEXT Q
4300 FOR Q=1 TO 50 : PRINT @ RND(1000), "YOU WIN" : NEXT Q : CLS
4400 PRINT @ 128, "HERE IS ANOTHER POINT FOR YOU"
4500 FOR Q=1 TO 500 : NEXT Q
4600 GOTO 1200

```

It is a game...put it on your computer and play it. This is guaranteed to improve your visualization of what is where on the pixel screen.

Here is another possibly useful technique.

Let us use our knowledge of graphics to create a bar graph or histogram for the number of students attending The University of Oklahoma.

```

100 CLS
110 PRINT @ 6, "GRAPH OF ENROLLMENTS AT THE UNIVERSITY OF OKLAHOMA"
120 REM LABEL THE LINES FOR THE BAR GRAPHS
130 PRINT @ 128, "1940"
140 PRINT @ 192, "1950"
150 PRINT @ 256, "1960"
160 PRINT @ 320, "1970"
170 PRINT @ 384, "1980"

200 REM DRAWS THE ACTUAL GRAPHS
220 FOR X=12 TO 42
230 SET(X,6) : SET(X,7)
240 NEXT X

320 FOR X=12 TO 56
330 SET(X,9) : SET(X,10)
340 NEXT X

420 FOR X=12 TO 64
430 SET(X,12) : SET(X,13)
440 NEXT X

520 FOR X=12 TO 102
530 SET(X,15) : SET(X,16)
540 NEXT X

620 FOR X=12 TO 124
630 SET(X,18) : SET(X,19)
640 NEXT X

```

A shorter and more powerful graph generator can be written using instructions we have not yet studied (see Lesson 13). In this version, the years and the values to be plotted are stored in DATA statements and entered into the computer using a READ instruction.

```

5 CLEAR 2000
10 CLS
100 PRINT @ 6, "GRAPH OF ENROLLMENTS AT THE UNIVERSITY OF OKLAHOMA"
110 READ Y, D
115 IF Y < 0 THEN 115
120 DATA 1940, 21, 1950, 28, 1960, 32, 1970, 51, 1980, 62, -1, -1
130 PRINT Y; STRING$(D,143)
140 GOTO 110

```

However, it is better to learn to walk steadily before one tries to run or to fly--We invite you to use the above program either with the given data or by changing instructions 100 and 120 to produce another histogram of more interest to you. The end of your data should be indicated by -1,-1 as above. If the data values for D lie between 1 and 59, the program will graph the data on a single line; otherwise more than one output line may be required in which case you may wish to insert 135 PRINT in the above program. Experiment a bit.

Try the small-flake snowstorm programs below.

```

100 SET(RND(127), RND(47))
110 IF RND(0)<.001 THEN CLS
120 GOTO 100

```

```

70 CLS
100 X=RND(127)
110 FOR Y=1 TO 43 + RND(3)
120 RESET(X,Y) : SET(X,Y+1)
130 NEXT Y
140 GOTO 100

```

Here is another interesting program. The students claim it is a "campus planning program".

```

50 N=1
80 CLS
100 PRINT "    CAMPUS PLANNING PROGRAM "; N
110 FOR K=1 TO 3+RND(17)

200 SX=RND(70)
205 SY=3+RND(23)
210 L=RND(55)
215 W=RND(20)
220 FOR X=SX TO SX+L
225 SET(X,SY) : SET(X,SY+W)
230 NEXT X
235 FOR Y=SY TO SY+W
240 SET(SX,Y) : SET(SX+L,Y)
245 NEXT Y
250 FOR Q=1 TO 100+RND(300) : NEXT Q
255 NEXT K
260 FOR Q=1 TO 800 : NEXT Q
265 N=N+1
270 GOTO 80

```

*If you'd like to have your N-S walls the same width as the E-W walls, add:  
243 SET(SX+1,V):SET(SX+L+1,V)*

*If you want a longer look at a special design on the screen, depress **SHIFT** **@** to halt the program.  
Then depress any key, without **SHIFT** to continue.*

Note instruction 260 FOR Q = 1 TO 800 : NEXT Q. This forces the program to count to 800 before it goes to instruction 80 and clears the screen. Separate instructions are separated by colons in lines 225, 240 250 and 260.

The following program contains several instructions we have not discussed yet. If you are particularly interested in learning about them,

consult Lessons 12 and 14. However, you can use the programs without having to understand `LEN(Z$)` or `MID$(Z$,K,1)`.

```

1000 CLS
110 Z$= "YOU HAVE ALREADY LEARNED LOTS, BUT STILL HAVE OTHER GOODIES
      TO INVESTIGATE, TOO."
120 K=0
130 E=INT((LEN(Z$)+1)/2)
140 FOR A=0 TO E STEP .5
150   X=15*SIN(A)
160   K=K+1
170   PRINT TAB(X+30); MID$(Z$,K,1)
180   FOR Q=1 TO 50 : NEXT Q
190 NEXT A
200 FOR Q=1 TO 300 : NEXT Q
210 GOTO 1000

```

Run the program. Then, change line 110 to

```
110 Z$ = "_____"
```

where you type any message you wish between the quotes.

Let us write a program to create a rectangle on the video screen.  
Select as vertices the following points:

A(25,10)	B(45,10)
C(25,30)	D(45,30)

We might believe that the result will be a square, since the distances AB, BD, DC, CA each appear to be twenty units in length, i.e.

X-distances are each  $45-25=20$  units  
Y-distances are each  $30-10=20$  units.

However, this is not valid. Let us write the program and then examine the output.

```

100 CLS
110 REM   DRAW LINE FROM (25,10) TO (45,10)
120 Y=10
130 FOR X=25 TO 45
140   SET(X,Y)
150 NEXT X

200 REM   DRAW LINE FROM (25,30) TO (45,30)
210 Y=30
220 FOR X=25 TO 45
230   SET(X,Y)
240 NEXT X

300 REM   DRAW LINE FROM (25,10) TO (25,30)
310 X=25
320 FOR Y=10 TO 30
330   SET(X,Y)
340 NEXT Y

400 REM   DRAW LINE FROM (45,10) TO (45,30)
410 X=45
420 FOR Y=10 TO 30
430   SET(X,Y)
440 NEXT Y

```

RUN the above program.

#### OUTPUT





The rectangle was not a square. Indeed, it is about twice as tall as it is wide, since the "spots" (pixels) located at (X,Y) are actually rectangles (as you can see by typing SET(100,30) ENTER). In other words, the X-direction units are only half as large as the Y-direction units.

To obtain a better approximation of a square, you could change each of the 45's to 65's (in instructions 120, 220 and 410). Change the steps as suggested and RUN the program again. It still isn't a perfect square, but it is much improved. What about lines 100, 200, and 400?

It is possible to make the lines of this square of equal weight by changing instructions 330 and 430 to

```
330 SET(X,Y) : SET(X-1,Y)
430 SET(X,Y) : SET(X+1,Y)
```

Try these alterations and see how the new program behaves.

It still does not produce a perfect square...perhaps because the rectangular "pixels" are really not exactly twice as high as they are wide.

Depress BREAK and CLEAR and type  
LIST ENTER

This will display your current program which should be:

---

```
100 CLS
110 REM      DRAW LINE FROM (25,10) TO (65,10)
110 Y=10
120 FOR X=25 TO 65
130   SET(X,Y)
140 NEXT X

200 REM      DRAW LINE FROM (25,30) TO (65,30)
210 Y=30
220 FOR X=25 TO 65
230   SET(X,Y)
240 NEXT X

300 REM      DRAW LINE FROM (25,10) TO (25,30)
310 X=25
320 FOR Y=10 TO 30
330   SET(X,Y) : SET(X-1,Y)
340 NEXT Y

400 REM      DRAW LINE FROM (65,10) TO (65,30)
410 X=65
420 FOR Y=10 TO 30
430   SET(X,Y) : SET(X+1,Y)
440 NEXT Y
```

Your task is to adjust the width of the rectangle to produce the best square you can. Increasing the 65's in instructions 120, 220 and 410 slightly (say, to 75) may be all that is needed. Play with it a bit yourself.

Actually, the program can be shortened considerably by drawing the two parallel lines in the same FOR.....NEXT loop, as illustrated below.

```
NEW
```

```
100 CLS
110 FOR X=24 TO 73
120 SET(X,10) : SET(X,30)
130 NEXT X
210 FOR Y=11 TO 29
220 SET(24,Y) : SET(25,Y)
230 SET(72,Y) : SET(73,Y)
240 NEXT Y
250 PRINT @ 405, "SQUARE" ;
300 GOTO 300
```

Instruction 250 prints the word SQUARE in the drawn square. The ; at the end of instruction 250 prevents that instruction from clearing the following line and thus disrupting your square.

Instruction

```
300 GOTO 300
```

puts the program in a "tightloop" that prevents the TRS-80 from displaying

```
READY
>
_
```

which could disrupt the display. To get the computer out of this tight-loop, depress the **BREAK** key.

CHR\$( )

Each screen character or action (including scroll, backspace, carriage return, and convert to double-size letters) has a numeric code that corresponds to it.

In Level II BASIC (but not in Level I) it is possible to use the instruction `CHR$(n)` (where *n* is an integer or a variable that has value between 0 and 255) to obtain the screen character or action corresponding to the number *n*.

```
100 INPUT K
110 PRINT K;
130 PRINT CHR$(K)
200 GOTO 100
```

will permit you to test this out.

Use the following input values, among others, to obtain a variety of symbols:

33, 36, 42, 60, 88, 91, 92, 109, 134, 148, 191

The following program will print the character number along with a row of that particular character. This may be useful as a border on a graphic display.

```
10 CLS
20 PRINT "PLEASE TYPE NUMBERS BETWEEN 33 AND 191"
100 INPUT K
110 PRINT K,
120 FOR L=1 TO 45
130 PRINT CHR$(K);
140 NEXT L
150 PRINT
200 GOTO 100
```

Try 33, 36, 38, 42, 47, 60, 62, 72, 88, 91, 92, 109, 125, 126, 134, 135, 141, 148, 152, 187 and 191 to obtain a variety.

To see the entire array of useful display symbols, change instruction 100 and add instruction 170 as suggested below:

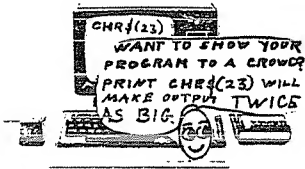
```
100 FOR K=33 TO 191
170 NEXT K
```

After you have run the program a few cycles, you may wish to permit *K* to range from 0 to 255 in instruction 100. But the effect of `CHR$( )` values outside of the range 33 to 191 is harder to visualize from the

program. For example: `CHR$(8)` backspaces and erases character.

The effect of `CHR$(W)` for several W-values is given below:

W-value	Effect or Character Printed
8	Backspaces and erases one character
10-13	Linefeed / Carriage Return
14	Turns on Cursor
15	Turns off Cursor
23	Converts to double-size letters



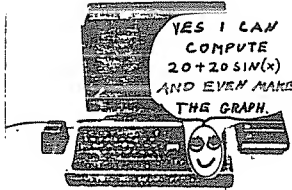
*This is handy in your programs!  
Just include*

*108 PRINT CHR\$(23)*

*and your output will be double-sized until the next CLS.*

24	Backspace Cursor ( + )
25	Advance Cursor ( + )
26	Downward Cursor (Linefeed or +)
27	Upward Cursor (Linefeed or +)
28	"HOME", Returns Cursor to (0,0) position
29	Moves Cursor to beginning of Current Line
30	Erase to End of Line
31	Clear to End of Frame
32	Space
33	!
34	"
35	#
36	\$
37	%

W-value	Effect or Character	W-value	Effect or Character
38	&	95	— (Underscore)
39	/	96-127	Lowercase a-z (but TRS-80 does not display lowercase)
40	(	128	Space
41	)	129-191	Various combinations of Rectangles for fast SET()
42	*	192-255	Tabs for 0 to 63 spaces
43	+		
44	,		
45	-		
46	.		
47	/		
48	0		
49	1		
50	2		
51	3		
52	4		
53	5		
54	6		
55	7		
56	8		
57	9		
58	:		
59	;		
60	<		
61	=		
62	>		
63	?		
64	@		
65-90	A - Z		
91	↑		
92	↓		
93	←		
94	→		

GRAPHING FUNCTIONS

Just so you won't feel this chapter is entirely frivolous, let's write a program to graph the function

$y = 20 + 20 \sin x$  for  $0 \leq x \leq 3.2$  (radians, of course). If you don't know what that means, don't let it bother you. The point is that the computer knows what it means.

If you just realize that for each value of  $x$  between 0 and 3.2 that you think of, the computer will supply you with a corresponding value of  $y = 20 + 20 \sin x$ , that's all you need to know. It doesn't really make any difference that  $\sin x$  is frequently used in trigonometry and you don't know any trigonometry...just hang on and have faith.

Let's write a short program to see that the computer really can supply the desired value.

Level II BASIC

```

90 PRINT "X", "Y"
100 FOR X=0 TO 3.2 STEP 0.1
110   Y=20+20*SIN(X)
120   PRINT X,Y
130 NEXT X
RUN

```

Your computer should produce:

X	Y
0	20
.1	21.9967
.2	23.9734
.3	25.9104
.4	27.7884
.5	29.5885
...	.....

Now, what we want to do is to graph this function instead of making a table of values.

Recall that the X,Y coordinates on the screen are (0,0) at the top left and Y increases downward. This is inverted from the usual graph.



We shall create a new variable  $Y1 = 40 - Y$  so that

Y	Y1
0	40
20	20
40	0
...	...

This will invert our graph. (Think about that a bit if it bothers you.)

We'll also introduce a new variable  $X1 = 10 * X$  to spread the function out on the screen. The program

```

90 CLS
100 FOR X=0 TO 12.5 STEP 0.1
120   Y=20+20*SIN(X)
130   X1=10*X
140   Y1=40-Y
150   SET(X1,Y1)
160 NEXT X
170 GOTO 170

```

will produce an acceptable graph of the function

$$y = 20 + 20 * \sin(x)$$

The last instruction 170 GOTO 170 is inserted to keep the TRS-80 from displaying

READY

> \_

which would spoil the graph. To get out of this "tight loop", depress **BREAK**.

Later, you can learn to make even fancier graphs that show the axes and units as well as the shape, but here we only introduce the basic idea.

The following program displays a series of interesting graphs.

```

100 CLS
110 A=1+RND(5)
120 B=1+RND(5)
130 C=1+RND(5)
140 PRINT A,B,C
150 FOR X=0 TO 12.7 STEP .1
160   Y=25-5*(SIN(A*X) - SIN(B*X) + COS(C*X))
170   SET(10*X,Y)
175   FOR Q=1 TO 20 : NEXT Q
180 NEXT X
190 FOR Q=1 TO 500 : NEXT Q
200 GOTO 100

```

```

10 REM  ANOTHER "RANDOM ART" PROGRAM
20 CLS
35 FOR L=1 TO 30
40   FOR K=0 TO 15
50     P=K*L4 + RND(64) -1
60     PRINT @ P, CHR$(128+RND(63));
70   NEXT K
80 NEXT L
90 FOR Q=1 TO 900 : NEXT Q
100 GOTO 20

```

After you have RUN the program a few times, add

```
115 PRINT @ RND(1023), CHR$(128+RND(63));
```

and RUN it again.



A ninth-grade student, Anthony Tipton, from Millwood High School, Oklahoma City, Oklahoma, produced the following rather interesting program after reading a preliminary version of this chapter.

```

3 REM  ANTHONY D. TIPTON
5 REM  1717 NE 50TH
7 REM  OKC,OK 73111
9 REM  427-0137
99 CLS
100 FOR Z=1 TO 3 : CLS
101   FOR X=0 TO 127
102     Y=23
103     SET(X,Y)
104   NEXT X
105   FOR Y=0 TO 47
106     X=63
107     SET(X,Y)
108   NEXT Y
109   FOR X= -6.2 TO 6.2 STEP .05
175     ON Z GOSUB 300, 400, 500
180     Y= -Y
190     X1=X*10 : Y1=Y*5
200     A=X1+63 : B=Y1+23
205     IF B<0 THEN 220
206     IF B>47 THEN 220
207     IF A>127 THEN 220
208     IF A<0 THEN 220
210     SET(A,B)
220   NEXT X
230 NEXT Z
250 GOTO 99

300 Y=SIN(X)
301 PRINT @ 896, "SINE";
302 RETURN

400 Y=TAN(X)
401 PRINT @ 896, "TANGENT";
402 RETURN

500 Y=X*X -4
501 PRINT @ 896, "PARABOLA";
502 RETURN

```

Sophisticated readers will see ways to speed up the program. However, it serves rather nicely at its current speed as a classroom demonstration illustrating how flat the sine curve is at max and min points, and how nearly linear where it crosses the x-axis, allowing for instructor comment on each curve.

SUMMARY OF LESSON 4

*Let's see what you have learned thus far...*

RND( $\emptyset$ )	Produces a random decimal value between 0 and 1, not including endpoints.
RND( $n$ )	Produces a random integer between 1 and $n$ , including endpoints.
PRINT @ 96 $\emptyset$ , "NEXT VALUE IS"; N	Prints at the bottom of the screen (location 96 $\emptyset$ ).
Run 1400 <span style="border: 1px solid black; padding: 0 2px;">ENTER</span>	Begins running a program at statement 1400.
SET(X,Y)	Turns on rectangle at point (X,Y).
RESET(X,Y)	Turns off rectangle at point (X,Y).
POINT(X,Y)	Tests point (X,Y) to see if it is lighted or not (very useful in some games).
FOR Q = 1 TO 400 : NEXT Q	The : separates different instructions on the same line. This particular line is a "time waster" to slow down the computer.
PRINT CHR\$( $n$ )	Obtains the character or screen action designated by $n$ .
ON N GOTO a, b, c,...,z	When N takes on the value $k$ , then program branches to $k^{\text{th}}$ statement number in the list a, b, c,...,z. (See Problem 11, Lesson 4.)

Mostly, we have learned to use some different "graphic" capabilities of the computer for fun and for work.

PRACTICE SESSION 4

1. If you haven't tried a variation of the PRINT AT RND(1000),"HI\_\_\_\_" program suggested, do so now. That is how you increase your understanding of the BASIC language you are using. Try this one, too.

```
NEW 
500 X=RND(125)
510 Y=RND(47)
520 SET(X,Y) : SET(127-X,Y)
530 IF RND(0)<.002 THEN CLS
540 GOTO 500
```

2. Try the "snowflake" program given in the text.

```
90 CLS
100 X=RND(127)
110 FOR Y=1 TO 43+RND(3)
120   RESET(X,Y) : SET(X,Y+1)
130 NEXT Y
140 GOTO 100
```

Then add

```
105 T=RND(127)
125 RESET(T,Y-1) : SET(T,Y)
```

Type LIST  to see the new program.  
Then, type RUN  to run it.

3. Add a third snowflake to the extended program of Problem 2 by adding appropriate instructions in 106 and 126. Try your program out.
4. Run the following program:

```
NEW 
100 X=0
110 Y=0
120 SET(X,Y) : SET(127-X,Y)
130 X=X+1
140 Y=Y+1
170 GOTO 120
```

If you are using Level I BASIC, it should fill the screen with diamonds.

If you are using Level II BASIC, the program will "bomb off" whenever Y>47 or X>127. This can be avoided by inserting:

```
150 IF Y > 45 THEN Y=RND(10)
160 IF X > 125 THEN X=RND(15)          Try it again!
```

5. Try the following program on your TRS-80.

NEW

```

90 CLS
100 X=4+RND(120)
110 Y=3+RND(40)
120 SET(X+1,Y+2) : SET(X,Y+2) : SET(X+1,Y+1) : SET(X,Y+1)
130 SET(X-1,Y) : SET(X-2,Y)
140 SET(X+2,Y) : SET(X+3,Y)
145 SET(X,Y-1) : SET(X+1,Y-1)
160 IF RND(0)<.02 THEN CLS
200 GOTO 100

```

After it runs, try changing 160:

```

160 IF RND(0)<.2 THEN CLS
and then to
160 IF RND(0)<.005 THEN CLS

```

Note: The programs of Problems 5 to 13 are somewhat related and may be placed on the TRS-80 without using NEW  to save re-typing.

6. Extend and modify the program of Problem 5 by adding:

```

170 IF RND(0)<.7 THEN 100
200 X=0
210 Y=0
220 SET(X,Y) : SET(127-X,Y)
230 X=X+1
235 IF X>120 THEN X=RND(6)
240 Y=Y+1
245 IF Y>45 THEN Y=RND(5)
270 IF RND(0)<.01 THEN 90
300 GOTO 220

```

Type RUN  . This will run the program from 200 on, to check it out. After it is checked out type RUN .

7. Run the following program:

```

1100 FOR X=32 TO 255
1110 PRINT X,
1120 FOR L=1 TO 45
1130 PRINT CHR$(X);
1140 NEXT L
1150 PRINT
1160 NEXT X

```

Run the program a couple of times. Then change statement 1100 to:

```
1100 FOR X=5 TO 35
```

and RUN the new program.

8. Try this program:

```
600 CLS
610 K=32+RND(150)
620 PRINT @ RND(1023), CHR$(K);
630 GOTO 610
```

After you have RUN the above program a few times, delete the semi-colon at the end of line 620 and RUN it again. Can you detect any difference? Why?

After RUNNING the revised program (without the ; ) add:

```
615 IF RND(0) < .01 THEN PRINT @ RND(1000), "HI your name ";
and RUN it.
```

9. Wasn't that fun! Now, try this one:

```
500 X=RND(125)
510 Y=RND(47)
520 SET(X,Y) : SET(127-X,Y) : SET(X,48-Y) : SET(127-X,48-Y)
530 IF RND(0) < .002 THEN CLS
535 GOTO 500
```

(Compare this with Problem 1.)

10. Here's another goodie.

#### FOR LEVEL II

```
700 FOR Q=1 TO 200 : NEXT Q : CLS
710 FOR X=15360 TO 16383
720   POKE X,191
730 NEXT X
760 RESET(RND(124)+1, RND(40)+2)
770 IF RND(0) < .001 THEN 700
780 GOTO 760
```

#### FOR LEVEL I

```
700 FOR Q=1 TO 200 : NEXT Q : CLS
710 FOR Y=0 TO 47
712   FOR X=0 TO 125 STEP 2
720     SET(X,Y) : SET(X+1,Y)
730   NEXT X
740 NEXT Y
760 RESET(RND(124)+1, RND(40)+2)
770 IF RND(0) < .001 THEN 700
780 GOTO 760
```

11. The following knits together the four separate programs which we tested in Problems 5, 6, 9, and 10. In each case, the individual programs have been modified to send control to instruction 800 every once in a while. Instruction 800 then sends control, at random, to one of the subprograms beginning at instruction 90, 100, 200, 500, 700, 200, 500, 100.

```

5 T=.01
10 DEFINT X,Y      (Omit this instruction on Level I BASIC.)
70 FOR Q=1 TO 200 : NEXT Q : CLS

100 X=4+RND(120)
110 Y=3+RND(40)
120 SET(X+1,Y+2) : SET(X,Y+2) : SET(X+1,Y+1) : SET(X,Y+1)
130 SET(X-1,Y) : SET(X-2,Y)
140 SET(X+2,Y) : SET(X+3,Y)
145 SET(X,Y-1) : SET(X+1,Y-1)
150 IF RND(0)<.02 THEN CLS
170 IF RND(0)<.7 THEN 100
180 IF RND(0)<T THEN 800

200 X=0
210 Y=0
220 SET(X,Y) : SET(127-X,Y)
230 X=X+1
235 IF X>120 THEN X=RND(6)
240 Y=Y+1
245 IF Y>45 THEN Y=RND(5)
295 IF RND(0)<T THEN 800
300 GOTO 220

500 X=RND(125)
510 Y=RND(47)
520 SET(X,Y) : SET(127-X,Y) : SET(X,48-Y) : SET(127-X,48-Y)
530 IF RND(0)<.002 THEN CLS
535 IF RND(0)<T THEN 800

700 FOR Q=1 TO 200 : NEXT Q : CLS
710 FOR X=15360 TO 15383
720 POKE X,191
730 NEXT X
760 RESET(RND(124)+1, RND(40)+2)
770 IF RND(0)<.0001 THEN 90
775 IF RND(0)<.1*T THEN CLS : GOTO 800
780 GOTO 760

800 ON RND(8) GOTO 90,100,200,500,700,200,500,100

```

On Level I BASIC use:

<pre> 710 FOR Y=0 TO 47 712 FOR X=0 TO 125 STEP 2 720 SET(X,Y) : SET(X+1,Y) 730 NEXT X 740 NEXT Y </pre>	}	<pre> 710 FOR Y=0 TO 47 712 FOR X=0 TO 125 STEP 2 720 SET(X,Y) : SET(X+1,Y) 730 NEXT X 740 NEXT Y </pre>
--	---	--

Instruction

800 ON RND(8) GOTO 90, 100, 200, 500, 700, 200, 500, 100

sends the program to the  $n^{\text{th}}$  instruction in the list 90, 100, 200, 500, 700, 200, 500, 100, depending upon the value of RND(8) which must be one of  $n = 1, 2, 3, 4, 5, 6, 7, 8$ .

12.

```

390 CLS
400 X=30+RND(65) : Y=10+RND(28)
410 SET(X,Y) : SET(126-X,Y)
411 SET(X,46-Y) : SET(126-X,46-Y)
415 X=X+RND(3)-2
420 X=X+RND(3)-2
430 IF X>125 THEN X= 125
435 IF X<1 THEN X=1
440 Y=Y+RND(3)-2
445 IF Y<1 THEN Y=1
450 IF Y>45 THEN Y=45
451 IF RND(0)<.009 THEN FOR Q=1 TO 500 : NEXT Q : CLS : GOTO 410
453 IF RND(0)<.01 THEN 400
455 IF POINT(X,Y) THEN RESET(X,Y) : RESET(126-X,Y) : GOTO 415
460 GOTO 410

```

Try the above program on your TRS-80. After you have enjoyed it a bit, undertake the following modifications. Is the design symmetric right to left? Is it symmetric top to bottom?

Add the following instruction:

```
413 FOR Q=1 TO 200 : NEXT Q
```

which does nothing but slow down the action by forcing the computer to count to 200 each time it passes instruction 413. Run the above a bit before continuing...so you can follow the action in slow motion.

Try changing the .01 in instruction 453 to .05 and see how the pattern is affected.

Can you combine the result of instructions 415 and 420 into one instruction? Do so. (Note: Although the instructions appear identical, the values of X on which they work are not, so be careful.)

13. Modify the program of Problem 11 to also include the "Art" program of Problem 12, or your modification thereof.
14. Write an ART program of your own.
15. Write a program using the PRINT CHR\$( ) instruction in some way.
16. Write a program to display an 8 by 8 chessboard (with black and white squares).
17. Write some programs of your own choice. You really do have the instructions needed for most programs.

18. Write a program that begins

```
1000 X=RND(50) : Y=RND(20) : L=RND(15)
```

and will then draw a square having upper left-hand corner (X,Y) and length of side L *y*-units. When your program is running well, enclose it in a FOR\_\_NEXT loop so the program draws from 6 to 10 squares. A possible over-program might be:

```
10 CLS
20 Z=RND(5)
30 FOR K=1 TO (5+Z)
1000 X=RND(50) : Y=RND(20) : L=RND(15)
```

*Your square-drawing program here.*

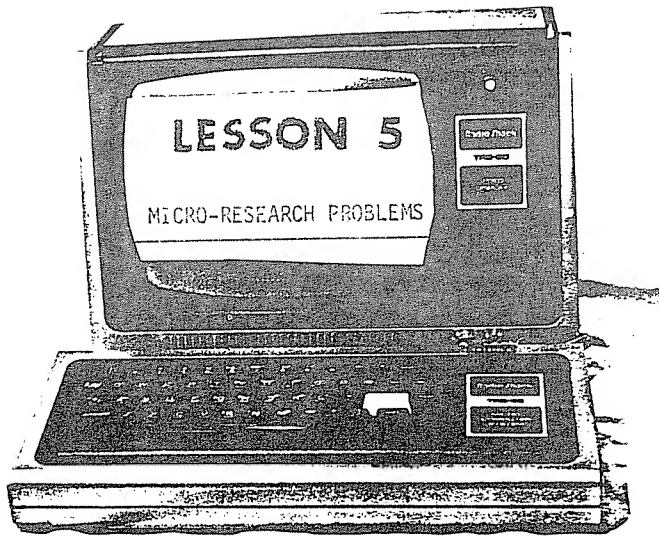
```
5000 NEXT K
5100 FOR Q=1 TO 5000 : NEXT Q
5200 GOTO 10
```

19. Write a program to draw a design of your choice: stairstep, circle, tree, box, face, robot, or whatever turns you on. However, choose your design first, then try to approximate it on the computer.
20. Make a bar graph showing data of your choice for several years. Note: scale your data so the longest bar ends before X = 125.
21. Make a cartoon of a dog that wags its tail on the display screen.
22. Make a smiling clown face on the display screen. Then expand your program so the clown winks or cries or talks or something that involves picture movement.
23. Create an animated "stick person" who walks across the screen, bows, and then continues walking offscreen.
24. Investigate some commercially available program that uses animation. Some possibilities are:

```
Dancing Demon
Life (Conway's simulation of cell growth)
Android Nim
Star Wars
Chess programs
```

See almost any of the magazines listed in Lesson 15 or consult your local computer store or Radio Shack.





### MICRO RESEARCH PROBLEMS

In Lesson 3, the number 30 Challenge Problem asks:

"The song 'Twelve Days of Christmas' mentions various gifts that 'my true love gave to me.' Let's interpret the song so that 'on the third day of Christmas, my true love gave to me

Three French hens  
Two turtle doves  
And a partridge in a pear tree.'

My true love gave me six presents (3 French hens, 2 turtle doves, and 1 partridge, if we don't count the pear tree as a gift). How many gifts in total did my true love give to me during the twelve days of Christmas? Write a program to determine the sum.

Answer: 364, if you do not count the pear tree as a gift."

A student submitted the following program in response to Challenge Problem 30. It does not produce the correct answer because the student has violated a very fundamental programming rule. Can you find the error and correct it?

```

5 CLS
10 S=0
20 FOR K= 1 TO 12
30   S=S+K
40   PRINT K;
50 IF K=1 THEN 90
60   K=K-1
70   GOTO 30
90   PRINT "    "; S
100 NEXT K
110 PRINT "    TOTAL="; S

```

Before you continue, see what you can do to help our slipshod student. .

The basic principle our slipshod student violated is.

Do NOT change a FOR...NEXT variable  
inside of the FOR...NEXT loop.

If you look at the output from the program on the previous page

1		1
2	1	4
2	1	7
2	1	10
2	1	13
2	1	16
		etc.

since the first column in each line (separated by ; spacing) presumably contains the K values and the right most values (separated by , spacing) give the current S values, it is apparent that K is not taking on the values from 1 to 12 as might be expected in FOR K= 1 TO 12.

Indeed, the program blunder (changing the value of K inside the FOR...NEXT loop) forces K back to K=1 by the end of each loop, so the new K value, namely K+1, is always K=2. Live and learn.

If you haven't worked on this Challenge Problem, you should before continuing. It is possible to make the "blunder" into a valid running program merely by changing instructions 20 and 100 and adding an instruction 25. See if you can do so before continuing.

Here is our new program, with new instructions at 20, 25, and 100 as indicated.

```

5  CLS
10  S=0
20  FOR L = 1 TO 12
25      K = L
30      S = S+K
40      PRINT K;
50  IF K = 1 THEN 90
60      K = K-1
70      GOTO 30
90  PRINT " ";S
100 NEXT L
110 PRINT "    TOTAL=";S

```

This produces the reasonably understandable output

RUN

```

1      1
2      1      4
3      2      1      10
4      3      2      1      20
5      4      3      2      1      35
6      5      4      3      2      1      56
7      6      5      4      3      2      1      84
8      7      6      5      4      3      2      1      120
9      8      7      6      5      4      3      2      1      165
10     9      8      7      6      5      4      3      2      1      220
11     10     9      8      7      6      5      4      3      2      1      286
12     11     10     9      8      7      6      5      4      3      2      1      364

```

TOTAL = 364

READY

>\_

Mathematicians might use the Greek symbol sigma,  $\Sigma$ , to indicate the sum and express this problem as

$$\sum_{L=1}^{12} \left( \sum_{K=1}^L K \right)$$

Would the program listed on the following page achieve the same result?

```

200 SL = 0
210 FOR L = 1 TO 12
220   SK = 0
230   FOR K = 1 TO L
240     PRINT K;
250     SK = SK + K
260   NEXT K
270   PRINT "   "; SK
280   SL = SL + SK
290 NEXT L
300 PRINT "      TOTAL="; SL

```

*Don't type NEW. Just start this program beyond where the first program ended, so you can run either one by using:*

RUN

or

RUN 200

or get both, by merely typing

RUN

The final total is the same in each case (TOTAL = 364) but the intermediate sums (at the extreme right of each line) are quite different.

Hmmm? How would you change the second program to get it to mimic the output of the first program? Well, we want line

270 PRINT " "; SK to print the value of SL instead of SK.

Let's change 270 to

270 PRINT " "; SL

and rerun it.

At first, that may look better, but a little more careful examination shows that this revision of the second program produces a running (cumulative) sum that is always one line late. Hmmm?

If we interchange instructions 270 and 280, giving

```

200 SL = 0
210 FOR L = 1 TO 12
220   SK = 0
230   FOR K = 1 TO L
240     PRINT K;
250     SK = SK + K
260   NEXT K
270   SL = SL + SK
280   PRINT "   "; SL
290 NEXT L
300 PRINT "      TOTAL="; SL

```



This program should now produce the same result as the program on the previous page. Does it? Yes, it does. Frankly, I prefer the program at

the top of the previous page that lists the number of gifts each day on the right edge, rather than the running total.

Actually, that problem is simple enough so that we could have computed the result without using a computer...but since a computer was readily available, it may have been easier to use the computer. Or it would have been, if we hadn't made so many foolish blunders in our program. However, correcting blunders is one way to learn to write blunder-free programs.

Let us turn our attention to some problems that are not easy to solve without a computer. The term micro-research problems seems appropriate for problems that are not to be found in most text books. The easiest way to find the answer to a micro-research problem may well be to solve it for yourself. Interestingly, you do have enough mathematical and computer programming ability, so that with the help of a TRS-80 micro computer, you can solve many problems that would have been beyond your ability and/or patience five lessons ago.

Let's get going!

Now that you can use your computer to solve a really difficult problem (obtaining roots of messy equations, Lesson 3) and have fun using the graphics capability of your TRS-80 (Lesson 4), it is time to think about some interesting micro-research problems on which you can expect to make reasonable progress using your computer. We shall walk through the solution of one quite difficult micro-research problem in detail - but you may wish to solve it yourself before reading on.

Problem:

Find perfect square integers like 5776 which have the property that they end in their square roots.  $\sqrt{5776} = 76$

Admittedly, this problem is not earthshaking, but it would not be very easy to solve without a computer. Try it yourself before continuing, with or without a computer.

## STEPS FOR COMPUTER-ASSISTED PROBLEM SOLVING

Successful problem solving is apt to involve several quite different stages. Here is a series of steps that is sincerely recommended.

1. Be sure you understand the problem. Try to restate it in several ways. Then try to find a general method (algorithm) for solving the problem.
2. Examine a simple special case first.
3. Then, if it seems appropriate, program a computer to examine another special case.
4. Modify your computer program to examine a different special case.
5. Modify and generalize your computer program. Include tests to be sure the program is working as expected.
6. Run the new program and examine the output.
7. Now re-examine the output. Use your common sense and mathematical acumen to see if you can devise a better (faster and/or safer) algorithm or prove a theorem that will help solve the problem, unless it is already solved.
8. Go back to step 5, if the original problem has not yet been solved to your satisfaction.

This seems rather round-about at first, but experienced computer problem-solvers have found it is a much better technique than the usual technique of the amateur, who tries to start by writing the final program, or at least with the program of steps 5 and 6 right away. This is fine on trivial problems, but on more difficult problems it is really better to start easy and work up gradually, testing each stage.

Try it on our problem.

1. Be sure you understand the problem. Try to restate it.

Original Problem

Find perfect square integers like 5776 which have the property that they end in their square roots.  $\sqrt{5776} = 76$

Restated Problem (same problem, but from another viewpoint )

Find non-negative integers  $N$  such that  $N^2$  ends in  $N$ .

Find integers  $N \geq 0$  such that their squares  $S = N^2$  end in the digits of  $N$ .

The first technique (algorithm) that suggests itself may well be just to examine  $S = N^2$  for each  $N$  and determine whether or not  $S = N^2$  ends in  $N$ . Later you may be able to devise a better algorithm.

2. Examine a simple special case first.

In attempting to solve any problem, it is well to examine several special cases before plunging in.

$N$	$\emptyset$	1	2	3	4	5	6	7	8	9
$N^2$	$\emptyset$	1	4	9	16	25	36	49	64	81

So the one-digit numbers  $N$  whose squares end in  $N$  are  $N = \emptyset, 1, 5, 6$ .

3. Write a computer program to examine another special case.

Let us write a program that will find the three-digit numbers  $N$  whose squares end in  $N$ . The last sentence displays two rather important problem-solving techniques:

1. In computing, it frequently pays to rephrase a question by turning it around a bit.
2. It is often helpful to examine a special case (here the 3-digit case) of a more general problem.

You may wish to try writing such a program yourself before continuing.

If you wish to examine the last 3 digits of a number  $S$ , then you need to devise a technique of finding the last 3 digits of a longer number. There is an easy way to do it using the  $\text{INT}(\quad)$  function.  $\text{INT}(\quad)$  produces the integer portion of the value inside of the parentheses. Thus,  $\text{INT}(34.761) = 34$ .

The instruction

$$T = S - 1000 * \text{INT}(S/1000)$$

will produce the last three digits of S and store them as T. If this is not obvious, try it for 2 or 3 integral values of S.

For example:  $S = 123456$   
 $S/1000 = 123.456$   
 $\text{INT}(S/1000) = 123$   
 $1000 * \text{INT}(S/1000) = 123000$   
 $S - 1000 * \text{INT}(S/1000) = 123456 - 123000 = 456$

Using this instruction, see if you can devise the desired program yourself.

A program to examine the 3-digit numbers N such that  $S = N*N$  ends in N:

```
100 FOR N = 101 TO 999
110   S = N*N
120   T = S - 1000*INT(S/1000)
130   IF N<T THEN 200
140   PRINT S;N
200 NEXT N
```

You may wish to run the above program before continuing.

The results are:

141376	376
390625	625

#### 4. Modify your program to examine other special cases.

Can you change the program to also obtain the 1-digit values of N whose squares end in N? — namely, 0,1,5,6 as we saw when we worked the problem by hand?

Of course, you can. Merely change instruction,

100 FOR N = 101 To 999	to	100 FOR N = 0 TO 9
120 T = S - 1000*INT(S/1000)		120 T = S - 10*INT(S/10)

and RUN the program.  
 Try it yourself.

What changes would you make in the program to obtain the 2-digit values of N such that  $S = N^2$  ends in the digits of N? Please do so before continuing.



Although our current attack is clumsy, even a clumsy attack is often better than no attack at all. Let us now try to produce a program which will do the modifications for us. This is unnecessary in this particular problem, since the modifications are easily done, but we may learn some useful tricks along the way. Examine the following program and be sure you see how it works before running it.

5. Generalize your computer program to examine several cases using the same program.

Our new fundamental program:

```

10 INPUT P
100 FOR N = 1+P/10 TO P-1
110   S=N*N
120   T=S-P*INT(S/P)
130   IF N<>T THEN 200
140   PRINT S;N
200 NEXT N

```

(where P is the integer 1, followed by as many zeros as there are digits in N)

The program seems to work well. If there is a way to change P from 10 to 100 to 1000 etc. as the program goes along, you could then use a slight modification of the old program. Try it yourself before continuing.

In any event, the modified program given below sounds as if it would work:

```

10 P=10
100 FOR N = 1 + P/10 TO P-1
110   S=N*N
120   T=S-P*INT(S/P)
130   IF N<>T THEN 200
140   PRINT S;N
200 NEXT N
210 P = P*10
220 GOTO 100

```

Try it...it works! Well, it works for a while anyway.

It really does need two more instructions

```

90 PRINT "NOW WORKING ON N<" ; P

```

Also change 220 GOTO 90.

This will tell you the general range of N-values on which the program is working. If it gets stuck, you at least know where. It will also give you estimates on how long it takes for each set of k-digit N values, for  $k = 1, 2, 3, \dots$ . Common sense suggests 3-digit numbers should take about 10 times as long as 2-digit numbers, since there are 10 times as many of them.

Remember to include another important check. Any computer has numbers  $S$  so large it cannot tell  $S$  from  $S+1$ . If a computer carries only six digits of accuracy, for example, and  $S$  is a 7-digit number, say

(True  $S$ ) = 7654321

The computer shows this as  $7.65432E+06$ , which is read  $7.65432 \times 10^6$  or 7654320. Now (True  $S+1$ ) = 7654322.

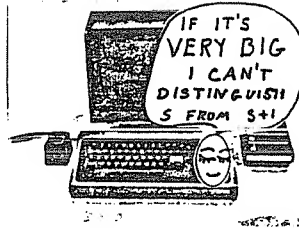
However, if your computer stores only six decimal digits of accuracy, then

$S+1 = 7.654322 \times 10^6$

which is chopped off (or possibly rounded) to six significant digits, giving

$7.65432E + 06$

which is precisely what you had for  $S$ . Thus, the computer cannot distinguish between  $S$  and  $S+1$ .



In this case, the final digit of  $S$  is no longer known. The computer substitutes zero for it, and cannot tell if  $S$  ends in the same digits as does  $N$ . Surely, you wish your program to STOP automatically if such a crisis should arise (and it does, inevitably ).

Do this by inserting the instruction

```
115 IF S=S+1 THEN PRINT "OVERFLOW ERROR ON S,N="; S;N : STOP
```

If the computer cannot distinguish between  $S$  and  $S+1$ , it will print the requested message and then STOP. Otherwise, the program will ignore instruction 115 and pass on to the next instruction.

The expanded program now reads:

```
10 P = 10
90 PRINT " NOW WORKING ON N< " ; P
100 FOR N = 1 + P/10 TO P-1
110 S = N*N
115 IF S = S+1 THEN PRINT "OVERFLOW ERROR ON S,N="; S;N : STOP
120 T = S-P*INT(S/P)
130 IF N<T THEN 200
140 PRINT S;N
200 NEXT N
210 P=P*10
220 GOTO 90
```

Let's return to "Steps for Computer-Assisted Problem Solving".

6. Run the general program and examine the output.

Give it a try on your own computer before continuing.

The output is:

LEVEL II BASIC .....	LEVEL I BASIC .....
NOW WORKING ON N<10	NOW WORKING ON N<10
25     5	25     5
36     6	36     6
NOW WORKING ON N<100	NOW WORKING ON N<100
625     25	625     25
5776     76	5776     76
NOW WORKING ON N<1000	NOW WORKING ON N<1000
141376     376	141376     376
390625     625	390625     625
NOW WORKING ON N<10000	NOW WORKING ON N<10000
OVERFLOW ERROR ON S,N = 3.35588E+07 5793	OVERFLOW ERROR ON S,N = 1.67772E+07 4096
BREAK IN 115	
<i>Notice, our program "bombed off" at P=10000 with S,N= 3.35588E+07, 5793 as a result of the test in instruc- tion 115.</i>	<i>Notice, our program "bombed off" at P=10000 with S,N = 1.67772E+07, 4096 as a result of the test in instruction 115.</i>

This seems to reach the limits of exploration using LEVEL I BASIC unless we are willing to construct our own special "multiple precision" routines.

It is time to consider a new concept in computing:  
DOUBLE PRECISION ARITHMETIC.

If you have LEVEL II BASIC on your TRS-80, you can also investigate the problem for 4-digit and 5-digit values on N (which produce 8 and 10-digit values of S) by including the instruction

5 DEFDBL S,P (which is read "Define Double".)

which produces double length (16-digit) values for any variable beginning with S or P. (Maybe we can even run the program for 6,7, and 8-digit N-values which have 12,14, and 16-digit squares — wait and see.)

However, troubles appear:

The TRS-80 does not produce a double length product if single length numbers are multiplied together. Thus  $S = N*N$  will not produce a valid 8-digit square, if N is a single precision 4-digit number

For  $N = 9376$  (TRUE  $S$ ) =  $N^2 = 87909376$

However, the computer rounds this to six digits and produces

$N*N = 8.79094E + 07$

which is stored in S as ~~87909400~~, rather than the correct value, ~~87909376~~.

The "obvious" solution is to make all three of S,P, and N double precision by using

5 DEFDBL S,P,N

However, the

FOR N = 1 + P/10 TO P-1

instruction will not accept a double precision variable for N.

This is easily fixed by using

100	N = 1+P/10		100	FOR N = 1+P/10 TO P-1
130	IF N<>T THEN 150	<u>in place of</u>	130	IF N<>T THEN 200
150	N = N+1			
200	IF N<P THEN 110		200	NEXT N

The program now reads:

```

5  DEFDBL S,P,N
10  P = 10
90  PRINT " NOW WORKING ON N<";P
100 N = 1 + P/10
110 S = N*N
115 IF S = S+1 THEN PRINT "OVERFLOW
    ERROR ON S,N="; S; N:STOP
120 T = S - P*INT(S/P)
130 IF N>T THEN 150
140 PRINT S;N
150 N = N + 1
200 IF N<P THEN 110
210 P = P*10
215 PRINT
220 GOTO 90

```

(Note+) COMMENTS

```

5  Omit instruction 5 if you are
    using LEVEL I BASIC.
10  The number of zeros in the
    current value of P is the num-
    ber of digits in N.
90  It is good practice in a pro-
    gram whose output may be scarce
    to include a statement display-
    ing what the program is doing.
115 This is a necessary precaution,
    when the computer cannot tell
    the difference between S and
    S+1, the program will STOP and
    so indicate on the screen.
210 Changes P to the next higher
    power of 10.
220 Repeats program with new P
    value.

```

The output is:

Level II BASIC

Level I BASIC  
(omitting instruction 5)

NOW WORKING ON N <10

```

25      5
36      6

```

NOW WORKING ON N< 10

```

25      5
36      6

```

NOW WORKING ON N <100

```

625     25
5776    76

```

NOW WORKING ON N <100

```

625     25
5776    76

```

NOW WORKING ON N <1000

```

141376   376
390625   625

```

NOW WORKING ON N <1000

```

141376   376
390625   625

```

NOW WORKING ON N <10000

87909376 9376

NOW WORKING ON N <10000

OVERFLOW ERROR ON S,N 1.67772E+07  
4096

NOW WORKING ON N <100000

8212890625 90625

NOW WORKING ON N <1000000

```

11963109376 109376
793212890625 890625  etc.

```

You will notice that for each new P value, it requires ten times as much computer time as for the previous P value.  
Run the program again and time it.

### Level II BASIC

Your program did not "bomb out" like Level I did. But it sure can use up lots and lots of computing time.

To get from

P=100 to P=1000 took about 1.5 minutes

P=1000 " P=10000 "" 15 minutes

P=10000 " P=100000 "" 2.5 hours

If you let your TRS-80 run another 24 or 48 hours, from P=100000 to P=1000000, you will also find

11963109376	109376
793212890625	890625

and who knows what else???

### Level I BASIC

You were stopped during

P=100000

at which time the instruction

```
115 IF S=S+1 THEN PRINT "OVERFLOW
      ERROR ON S,N=";S;N:STOP
```

halted the run. You may wish to delete instruction 115 and rerun the program to see what happens.

Remember you can always stop the program by depressing **BREAK**. You may then learn the values it was working on by typing

```
PRINT P,N,S,S+1
```

**ENTER**

NOW IS THE TIME FOR US TO GET SMART.

This program really works the problem in a very crude brute-force way. We are just examining every number, N. The number of numbers to be examined increases sharply as P increases.

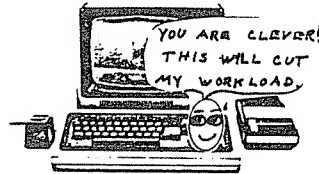
	<u>No. of cases</u>	<u>Approximate time</u>
1 < P < 10	8	Less than 1 second
10 < P < 100	88	9 sec
100 < P < 1000	898	90 sec = 1.5 min.
1000 < P < 10000	8998	15 min.
10000 < P < 100000	89998	150 min = 2.5hr.
100000 < P < 1000000	899998	24 hr.
1000000 < P < 10000000	8999998	9 or 10 days

Clearly we can reasonably expect to examine P < 10, 100, 1000, 10000, and even 100000, but after that, the computer time involved becomes exorbitant. Of course, you may reason that the TRS-80 might just as well be working on your problem as be sitting idle. It takes very little power. (It uses about as much power as a small light bulb, if you turn off the CRT display tube, which you might as well do if you are not watching it. You can turn it on and recapture whatever would have been on the screen at that moment. The computer continues to compute even when the display tube is turned off.)

Returning to our "Steps for Computer-Assisted Problem Solving",

7. Re-examine the output. Use your common sense and mathematical knowledge to devise a better (faster) algorithm to help solve the problem.

We really should have recognized at the beginning that  $N^2$  must end in 0, 1, 5, or 6 if  $N^2$  ends in  $N$ . This would have saved almost 60% of our computing time. Actually if  $N > 9$ , then  $N$  cannot end in 0 since then  $N^2$  would end in 00, etc. Similarly if  $N > 9$ ,  $N$  cannot end in  $k1$  since then  $N^2$  would end in  $(2k)1$  since  $(10k+1)^2 = 100k^2 + 20k + 1$ . Similar mathematical arguments show that if  $N > 9$  and if  $N^2$  ends in  $N$ , then  $N$  ends in 5 or 6. So instead of increasing  $N$  by 1 each time we can jump by 10 and test both  $N$  and  $(N+1)$  to see if either has the desired property. The following program will cut 80% of our running time--doing an hour's testing in 12 minutes.



```

5  DEFDBL S,P,N
10  P=10
90  PRINT "NOW WORKING ON N<";P
100  N = 5 + INT(P/100)*10
110  S = N*N
115  IF S = S + 1 THEN PRINT "OVERFLOW ERROR ON S, N =" S;N:STOP
120  T = S - P*INT(S/P)
130  IF N < T THEN 150
140  PRINT S;N
150  N1 = N+1
152  S1 = N1*N1
154  T1 = S1 - P*INT(S1/P)
156  IF N1 < T1 THEN 160
158  PRINT S1;N1
160  N=N+10
200  IF N < P THEN 110
210  P = 10*P
215  PRINT
220  GOTO 90

```

Try it. It works, and we really might have been smart enough to have thought of discarding 80% of our unsuccessful cases in the beginning.

However—now that we have some output to look at, it may be possible to find an even more clever way to speed up our program. Look at the output. Do you notice anything special?

$N^2$	N
25	5
36	6
625	25
5776	76
141376	376
390625	625
87909376	9376
8212890625	90625
11963109376	109376
793212890625	890625

Two items demand our attention:

I. When we investigated the one-digit N's by hand and on the first program (modified to numbers from 0 to 9), we found

$N^2$	N
0	0
1	1
25	5
36	6

but our current computer program missed the first two values 0, 1.

Just why did that happen, and do you think that it missed any other values between  $P = 10$  and  $P = 1000000$ ? Are You Sure?

We did obtain all the values between  $P = 10$  and  $P = 1000000$ , but did miss  $N = 0$  and 1 because this program did not test  $N = 0$  or  $N = 1$ , since it started at  $N = P/10 + 1$  for  $P = 10$  which is 2.

If you didn't notice that earlier, you need more computing experience before you can be considered a qualified amateur.

II. Notice: The only large values of N such that  $N^2$  ended in N were values of N that themselves ended in those exact digits for which we found some previous satisfactory N. Now that may be a real clue!



Consider the case starting at  $P=1000$ . We found

$N^2$	$N$
141376	376
390625	625

It seems reasonable to believe that any 4-digit value of  $N$  whose square ends in  $N$  will have the value of  $N$  itself ending in either 376 or 625 since otherwise,  $N^2$  would not end in  $N$ . Hmm. Well, yes -- that seems reasonable.

However, some things that "seem reasonable" turn out not to be true. For example, one of my students extended the above conjecture by asserting, "Since there is only one 4-digit value of  $N$  such that  $N^2$  ends in  $N$ , there can be only one such  $N$  having  $k$  digits for any  $k>4$ ."

However,

$$N = 109376$$

$$N^2 = 11963109376$$

$$N = 890625$$

$$N^2 = 793212890625$$

blows that "reasonable conjecture".

(Why?) Well, because the student didn't recognize that the six-digit 109376 was a suitable extension of the four-digit  $N = 9376$  under our original guess, with 0 as the fifth digit. Hmm.

Well, how about the original guess? If we could show that

Conjecture

If a given  $N^2$  ends in the digits  $N \geq 10$ , then  $N$  itself must end in the same digits as some smaller such  $N$ .

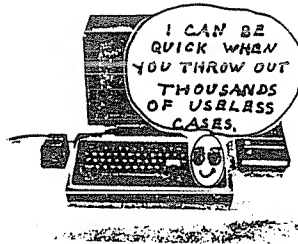
Not yet proved.

Would it be helpful in our search?

Well, you just bet it would be!

For example, in examining the 5-digit  $N$  values instead of examining 89998 cases, we would only need to examine the 20 cases X9376, X0625 (and possibly the 10 more cases X0376 unless we can show that is also impossible.... which we can).  
I'm not going to prove the conjecture--but it is true.

To cut down from 89998 cases to 20 cases for the 5-digit  $N$  values is a savings of 99.98% of the numbers to be tested. The 6-digit values produce even greater savings. What would formerly have taken twenty-five hours should now run in less than a minute, even if each number evaluated took several times as long to test (since the program may be messier than before). That is indeed a worthwhile savings.



For now let us assume the slightly stronger result:

If  $N^2$  ends in  $N$ , then  $N$  ends in the digits of some smaller  $K$  such that  $K^2$  ends in  $K$ . Furthermore, the previous  $K$  will be a  $K$  with one fewer digit than  $N$ , with the possibility of leading zeros on a smaller  $K$  such as 90625 being the  $N$  for  $K = 0625$ ,  $K^2 = 39025$ .

Let's start our program with 3-digit  $N$  values. This will enable us to test the program and see if it is running as we believe it should.

Store the "last two successful values of  $N$ " that we are trying to extend in locations A and B. Starting with the 3-digit values of  $N$ , there will be

P = 1000  
 A = 25 (since when  $N = 25$ ,  $N^2 = 625$ )  
 B = 76 (since when  $N = 76$ ,  $N^2 = 5776$ )

to properly "seed" our program with the successful 2-digit values to be extended.

Continuing with "Steps for Computer-Assisted Problem Solving," we find:

8. Go back to step 5.
5. Modify and generalize your computer program. Include tests to be sure the program is working as expected.

6. Run the new program and examine the output.

The following program seems to produce the desired results in moderate time. Study it until you understand it well enough to explain it to one of your classmates who needs help.

```

5 DEFDBL S,P,N,A,B           (Omit 5 if using LEVEL I BASIC.)
10 P=1000
20 A=25
30 B=76

90 PRINT "      WORKING ON N < " ; P
95 Q=P/10

99 REM      THIS BLOCK EXTENDS A VALUE
100 FOR K=1 TO 9
105 N = K*Q + A
110 S = N*N
115 IF S=S+1 THEN PRINT "OVERFLOW ERROR ON S,N = ";S;N: STOP
120 T = S - P*INT(S/P)
130 IF N<>T THEN 200
140 PRINT S;N
150 A=N
160 GOTO 300
200 NEXT K
210 REM      END OF BLOCK THAT EXTENDS A VALUE

250 REM

299 REM      THIS BLOCK EXTENDS B VALUE
300 FOR K=1 TO 9
305 N = K*Q + B
310 S = N*N
320 T = S - P*INT(S/P)
330 IF N<>T THEN 400
340 PRINT S;N
350 B=N
360 GOTO 500
400 NEXT K
401 REM      END OF BLOCK THAT EXTENDS B VALUE

450 REM

500 P = P*10
520 GOTO 90

```

Put it on your computer and see how it runs.

Output from Level II BASIC

WORKING ON N < 1000  
 390625 625  
 141376 376

WORKING ON N < 10000  
 87909376 9376

WORKING ON N < 100000  
 8212890625 90625

WORKING ON N < 1000000  
 793212890625 890625  
 11963109376 109376

WORKING ON N < 10000000  
 8355712890625 2890625  
 50543227109376 7109376

WORKING ON N < 100000000  
 166168212890625 12890625  
 7588043387109376 87109376

WORKING ON N < 1000000000  
 OVERFLOW ERROR ON S,N = 1.704786682128906D+17 412890625

BREAK IN 115

READY

>\_

Output from Level I BASIC

WORKING ON N < 1000  
 390625 625  
 141376 376

WORKING ON N < 10000  
 OVERFLOW ERROR ON S,N = 2.13906E + 07 4625

Now we need to ask if the original problem has been solved.

If it was to find all perfect squares  $N^2$  such that  $N^2$  ends in the N, then it is only partially solved. The program found all such  $N < 412890625$  but there is no reason to suspect that is all there are. There is also the possibility our program may be defective and may have overlooked some N values less than  $10^9$  (After all - we did overlook 0 and 1 - remember?).

This is one way research is done. First, investigate a few special cases. Next, use a simple program to investigate a few more cases. If the results seem worthwhile, automate your program and let it run a bit. Next, examine the output, make some conjectures and prove or disprove

their validity. Then, devise a new technique (algorithm) for carrying out the investigation and implement it.

Now you are ready to try to prove the most general case. It may or may not yield to your effort - but in any case, you have a pocketful of results that were previously unknown. We shall not attempt to go further here, but strongly suspect some able student will do so.

There is one additional check that should be made.

Did our last two programs produce the same results as far as they overlapped? If not, why not? This is important.

The two programs are:

<u>Program from page 89</u>	<u>Faster program from page 93</u>
5 DEFDBL S,P,N	5 DEFDBL S,P,N,A,B
10 P = 10	10 P = 1000
90 PRINT "NOW WORKING ON N<";P	20 A = 25
100 N = 1 + P/10	30 B = 76
110 S = N*N	
115 IF S = S+1 THEN PRINT "OVERFLOW ERROR ON S,N=";S;N:STOP	90 PRINT " WORKING ON N<";P
120 T = S - P*INT(S/P)	95 Q=P/10
130 IF N<>T THEN 150	99 REM THIS BLOCK EXTENDS A VALUE
140 PRINT S;N	100 FOR K=1 TO 9
150 N = N + 1	105 N = K*Q + A
200 IF N<P THEN 110	110 S = N*N
210 P = P*10	115 IF S=S+1 THEN PRINT "OVERFLOW ERROR ON S,N=";S;N:STOP
215 PRINT	120 T = S - P*INT(S/P)
220 GOTO 90	130 IF N<>T THEN 200
	140 PRINT S;N
RUN	150 A=N
	160 GOTO 300
	200 NEXT K
	210 REM END OF BLOCK THAT EXTENDS A VALUE
	250 REM
	299 REM THIS BLOCK EXTENDS B VALUE
	300 FOR K=1 TO 9
	305 N = K*Q + B
	310 S = N*N
	320 T = S - P*INT(S/P)
	330 IF N<>T THEN 400
	340 PRINT S;N
	350 B = N
	360 GOTO 500
	400 NEXT K
	401 REM END OF BLOCK THAT EXTENDS B VALUE
	450 REM
	500 P = P*10
	520 GOTO 90
	RUN

The two rather different programs seem to agree as far as they go, but the longer program produces more results in two minutes, than the old program did in 48 hours or even a week. That is what is meant by effective programming.

Actually all values of  $N$  such that  $N^2$  ends in  $N$  for  $N$  of 100 or fewer digits are known, but their investigation uses more mathematical sophistication than we are willing to impose here.



5-22

#### WHAT WE HAVE LEARNED IN LESSON 5.

The most important thing in Lesson 5 is the realization that you do already have enough computer adroitness to successfully undertake the investigation (and at least partial solution) of problems so difficult that a month ago you wouldn't even have tried them. You should also have discovered that it is frequently a nontrivial (but perfectly possible) task to program a computer to help solve research level problems.

Most important of all - you have discovered one of the fundamental truths of computer programming.

When you have a program written and it is debugged and running properly, then if you reflect on the problem you will frequently find a better way to do it. If the problem is major, this is the time to think hard, then scrap your earlier effort and start over.

Of course if the problem is a "one time only" problem and if your existing program produces the desired results while you are thinking, you can forget the reprogramming unless you want to show the program to someone else or to use it again later.

SOME MICRO-RESEARCH PROBLEMS  
YOU MAY UNDERTAKE

Don't expect all of these problems to be easy, or even solvable with your present knowledge. They are micro-research problems for your consideration. If you do more than two problems, you are to be congratulated. After working on a problem as stated, see what extensions and modifications you can devise; work on them.

1. If the 6's are cancelled in  $16/64 \div 16/64 = 1/4$  the result is correct. ( $26/65 \div 26/65 = 2/5$  is another example.) Find all examples of proper fractions such that  $AB/BC = A/C$  with  $A \neq B$  where A,B,C are digits between 0 and 9 inclusive.
2. If  $N! = 1*2*3*\dots*(N-1)*N$  for  $N \geq 1$   
and  $S(K) = 1! + 2! + 3! + 4! + \dots + K! = \sum_{N=1}^K (N!)$ ,  
for what values of  $K \geq 1$  is  $S(K)$  a perfect square?  
(This requires some easy mathematical thinking after you have written and run a computer program.)
3. Write a program to accept an integer  $N > 2$  as input and then type out N and the factors of N.
4. Let your input be three positive numbers A,B,C (not necessarily integers).
  - a) Determine whether or not A,B,C can be the sides of a triangle. (Note: If A,B,C = 1,5,3, the answer is "no".)
  - b) If A,B,C are acceptable as sides of a triangle, extend your program to also print the area of the triangle.
  - c) If A,B,C do determine a triangle, extend your program to also compute the angles of the triangle.
5. a) Write a program that will accept a positive integer N as input and determine whether or not N is a prime. (Note: 1 is not a prime. An integer  $N > 1$  is prime if its only positive factors are 1 and N.) Compare your program with those written by your classmates. Test it for several values of  $N > 60000$ . See whose runs fastest, and why.
  - b) Make a table of primes  $< 50000$ .
  - c) Extend your program to print out all the prime factors of N.
6. Write a program to accept two positive integers M and N and to print out M, N and also the largest positive integer which divides both M and N. Such an integer is called the greatest common divisor and may be found using a technique called Euclid's Algorithm or by simply trying all positive integers from the smaller of M and N down to 1 until you find a value which will divide both.

If you use the latter (inefficient, but effective) technique, your program could begin:

```

100 INPUT "M,N =" M,N
106 IF M<=0 THEN 100
108 IF N<=0 THEN 100
109 REM: INSTRUCTION 110 SETS S = SMALLER OF (M,N) FOR USE IN 200
110 IF M>N THEN S = N ELSE S = M
200 FOR D=S TO 1 STEP -1
.
.
.

```

7. Find two consecutive integers whose squares each use exactly the same digits (in a different order, of course). Thirteen and fourteen are such a pair;  $13^2 = 169$  uses the same digits as  $14^2 = 196$ . Similarly,  $157^2 = 24649$  and  $158^2 = 24964$ . Find other pairs of consecutive integers whose squares are composed of the same digits.

8. a) Write a computer program to accept three values M,D,Y and determine whether or not they are acceptable values for Month, Day, Year in that order. If  $Y < 1590$  you may prefer to state the date is unacceptable rather than consider the various calendar reforms that took place prior to 1590.  
 b) Extend the above program to accept two sets of three numbers and if both are acceptable as Month, Date, Year values, determine the number of days between them including both dates in your count.  
 Example: 2,20,1978 to 2,22,1978 is 3 days.

9. For the distinct digits A, B, C, D, E, F, G, H, I, J the product

$$A*(BCDE) = FGHIJ$$

has solution  $4*(7039) = 28156$  and a dozen other solutions. Find them.

10. Write a program to accept a positive integer K and print it out and also print out the integer obtained by reversing the digits of K.  
 Example: If K = 12345,  
 print out: 12345                      54321

11. The sequence  
 1 2 3 4 6 8 9 12 16 18 24 27 32 ...

is made by listing numbers of the form  $N = 2^k \cdot 3^m$  (with k and m integers  $k \geq 0, m \geq 0$ ) in order of increasing size. It is not easy to devise an algorithm to generate and print them in increasing order, but that is only part of your task. The even more difficult problem is to find the 1000th term of the sequence, or more generally, given an integer T, find the T-th term of the sequence. (Thanks to Fred Gruenberger for suggesting this problem.)



12. Write a program that will draw a random maze on the screen such that it is possible to reach every cell of the maze, but there is only one "best" path (i.e. without retracing) through the maze. Then, extend your program so that the computer will "solve" the maze it created.
13. a) Determine the sum of the N-digit numbers that can be formed using N specified digits for each possible set of 2,3,4 and 5 digits.  
Examples:
- N = 2   Digits = 3,8   Then sum = 38 + 83 = 121  
              Digits = 4,1   Then sum = 14 + 41 = 55   (*Yes, we count*  
              Digits = 2,2   Then sum = 22 + 22 = 44   *22 both ways.*)
- N = 3   Digits = 1,2,3   Then sum =  
                              123 + 231 + 312 + 132 + 321 + 213 = 1332  
              Digits = 3,8,9   Then sum =  
                              389 + 893 + 938 + 983 + 839 + 398 = 4440
- b) Notice that the sum seems always to be divisible by 11. Can you prove this in general? What other factors will the sum have?
- c) Can you show that an N-digit number will have N! different permutations, (N-1)! of which start with each digit, and (N-1)! of which have each digit in the second place, etc.?
- d) If you prove each of the above, you may also be able to show that the sum of the N! numbers obtained as permutations of the N digits will always be (N-1)! \* (sum of the original digits) \* (the integer consisting of N ones). For N = 8, Digits = 1,2,3,4,5,6,7,8 there are 40320 integers and their sum is 2015999979840.
- e) What happens if a number containing repeated digits is counted only once? Thus N = 2, Digits = 2,2 Sum = 22 (not 22 + 22 = 44). [This is a rough problem.]
14. A palindrome is a phrase or sentence which reads the same forward as backward.

"Too Hot To Hoot"  
 "Ma Is As Selfless As I Am"  
 "Retracting, I Sign It, Carter"  
 "Marge Lets Norah See Sharon's Telegram"

There are also numbers which are palindromes. It is fairly easy to find perfect squares which are also palindromes.

$$(111)^2 = 12321$$

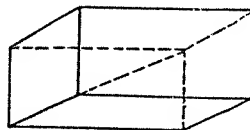
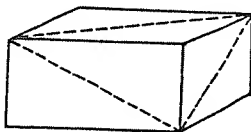
$$(202)^2 = 40804$$

$$(264)^2 = 69696$$

It is possible to prove that there exist infinitely many perfect squares that are palindromes. However, perfect squares that are palindromes having an even number of digits such as  $(836)^2 = 698896$  are fairly rare.

Investigate the subject of perfect squares that are also palindromes.

15. For what values of  $N > 1$  does  $W = 2^N - 1$  contain a factor  $> 1$  which is a perfect square?
- If  $N = 6$ ,  $W = 2^6 - 1 = 63 = 3^2 \cdot 7$
- If  $N = 7$ ,  $W = 2^7 - 1 = 127$  which does not contain a square factor  $> 1$
- If  $N = 8$ ,  $W = 2^8 - 1 = 255$  which does not contain a square factor  $> 1$
16. a) It is easy to find Pythagorean rectangles having integral sides and integral diagonals  $(l, w, d) = (3, 4, 5)$  or  $(5, 12, 13)$ . But can you find a box (rectangular parallelepiped) having three distinct integers as dimensions, each of whose face diagonals is also an integer?
- b) If so, can you also find one whose body diagonal as well as face diagonals are integers?



17. Use your TRS-80 to design some personal greeting or Christmas cards. Take photographs of the TRS-80 showing the design on the screen and enough of the keyboard to be recognizable. Send one to your authors.

# PART II

You are ready to proceed on your own now. No one is expected to work all of the research problems in Lesson 5, but you should work several and at least read and consider the others. The way to learn computing is to compute, and then reflect.

The remainder of this book discusses useful techniques and additional BASIC instructions you may or may not need to know about. Lessons 6 to 17 may be read in any order desired, or ignored. Read about these techniques when you are ready to use them. You already have skill enough to solve many problems using the computer as a tool. We suggest you glance briefly at the summary below to ascertain the general content of the lessons beyond 5. Then read them when the need for such specialized information is felt. Read whatever interests you most, and continue with the micro-research problems presented in Lessons 5 and 17.

Lesson 6 is devoted to the use of the TAPE CASSETTE for storage of programs and data.

Lesson 7 is devoted to GAMES and TAPED PROGRAMS you may enjoy.

Lesson 8 discusses the EDIT instruction in greater detail.

Lesson 9 is an introduction to simulation, one of the most important uses of computers in today's world.

Lesson 10 discusses double precision variables (introduced in Lesson 5) before introducing subscripted variables and the fascinating (and vital) string variables (more details in Lesson 14). The lesson closes with arrays (matrices) and how to use them.

Lesson 11 is a collection of tips that may help you overcome or avoid difficulties. Remarks on saving memory space and computing time as well as a listing of the meanings of major error codes are presented in this lesson.

Lesson 12 Although the PRINT instructions already at your disposal are sufficient for most users, your TRS-80 Level II BASIC has a variety of other PRINT instructions and special commands that are frequently useful:

```
PRINT TAB( )      PRINT STRING$(K,"#")
PRINT USING A$,K  PRINT CHR$(n)
```

Lesson 13 Lesson 4 may meet all your graphic needs, but Lesson 13 explains the pixel notation and introduces several time-saving techniques and instructions.

Lesson 14 extends your knowledge of string variables, and introduces elementary conversational programs in which the computer simulates conversation. The ROBOT COUNSELOR program and the CIPHER program may also interest you.

Lesson 15 explains how and where to look for additional information on computing with particular emphasis on the TRS-80 Level II BASIC. Several computer related journals are listed with brief annotations.

Lesson 16 discusses several BASIC instructions not used earlier in in this book.

Lesson 17 extends the collection of seventy-five microRESEARCH PROBLEMS. The way to learn computing is to compute and then to examine your programs and results to see if you could have devised a better program.

The discussions presented Lessons 5 and 17 are planned to help you develop efficient as well as effective programs and to whet your problem solving ability. Many of the problems presented are readily extended. Their solutions frequently suggest further explorations worthy of your talent. This is why they merit the adjective "RESEARCH". The wide variety of problems includes something for every taste: several arithmetic explorations, greeting card graphics, Haiku poetry, logical decisions, calendar problems, dart and target games, loan and finance programs, treasure hunt games, caricatures, speech timer, puzzles, recursive functions, polygonal maps, a medical emergency prompter, number theory problems, lattice problems, magic prime squares, graphs, monkeys at typewriters, amicable and sociable numbers, dance partners and the eight queens problem.

THE IMPORTANT THING IS TO HAVE FUN WITH YOUR micro COMPUTER



TAPE CASSETTE for programs and data storage.

Occasionally you have a special program worth saving for future use or to show off to a friend. Put it on tape. Tape is also handy for storing data.

Each TRS-80 comes with a tape recorder for storing programs and data that you wish to save. It uses regular (high quality) tape cassettes. I personally prefer the short 30 minute (15 minutes per side) tapes rather than the 45 minute, 60 minute or 90 minute tapes since the tape recorder starts, stops and operates more uniformly when the reel contains less tape. Try it and suit yourself. Radio Shack sells a special 10-minute tape for use on TRS-80 which has no "leader" on it--but any high quality 30 minute (15 minutes per side) tape will serve--just so you don't try to record on the leader.

*For readers who have older CTR-40 and CTR-41 tape recorders:*

The original tape recorders supplied with TRS-80s were CTR-41's on which the REM jack had to be unplugged before the tape was repositioned. The CTR-41 also requires the use of a "dummy plug" in the MIC jack at all times. The CTR-80 now furnished should not have a dummy plug inserted at any time.

Instead of unplugging the REM jack on the CTR-41 you can use a short program.

```
9000 OUT 255,4
9010 INPUT "REPOSITION TAPE, PRESS (ENTER) WHEN READY" ; A$
9020 END
```

Then type RUN 9000 when you wish to reposition the tape.

*For readers with pre-1980 CTR-80 tape recorders:*

The more recent CTR-80's can use "fast-forward" or "rewind" without unplugging the connection between TRS-80 and the cassette at the REM jack.

This is much nicer.

In either case the volume control was critical in getting usable programs and data back into the TRS-80 -- on early (pre-July, 1979) TRS-80s.

The following may be helpful:

Radio Shack announced (Spring, 1979) a "fix" that they will install free to make the recording level less sensitive on existing TRS-80's. Current TRS-80's already have it installed. Your TRS-80 Level II keyboard has a catalogue number on the underside of the case. If the catalogue number ends -1 (example 26-1006-1), the modification has already been made. If not, it will be worthwhile to take the keyboard unit to Radio Shack to have the free "fix" made. Cassette storage is much easier to use thereafter.

If your CTR-80 cassette player was manufactured before February, 1979 (see date stamped on cassette box--Feb., 1979 = 2A9) take the cassette player to your Radio Shack store, too. They will install a small capacitor in the innards of the electronics to avoid a possible "noise spike" on your recorded tape if you push the stop button while it is reading tape, an infrequent but annoying happening. If the date of manufacture is Feb., 1979 or later (say 6A9 or 1A0) this extra protection has already been installed at the factory. We recommend setting modified CTR-80 cassette recording level at 4 both for recording and playback. Once you find the best volume for your system, a drop of white correction fluid (Snopake, Liquid Paper, etc.) or paint will mark the location for future reference.


In any case, the usual tape care is essential.


1. Don't leave cassettes near a magnetic flux source like your power supply or near a magnet as in a motor or a speaker. Also, don't expose cassettes to extreme heat or cold.
2. Keep your cassette capstan and rollers clean. Dirty heads cause lots of troubles. If you don't know how to clean them, your nearby Radio Shack or tape recorder store will be glad to show you, and sell you a cleaning kit (<\$2).

To load a tape that already contains a program

Plug in everything carefully.

110 volt (Don't expect to use the batteries.) black power cord

DIN plug to TRS-80 keyboard hole labeled TAPE with 

pip on plug in slot on socket ,

3 plugs on other end of cable connect to cassette player

BLACK plug into EAR hole

LARGE GREY plug into AUX hole

SMALL GREY plug into small MIC hole (CTR-80)  
or REM hole (CTR-41)

If you are using CTR-41 put dummy plug into MIC and leave it there.

If you are using CTR-80 do not use dummy plug--just leave big MIC hole empty.

NEXT:

Adjust the position counter back to zero. (They register differently on CTR-41 and CTR-80, so be a bit suspicious.)

Check the volume adjustment. (This is fussy, experiment a bit.)

CTR-41 5 to 8

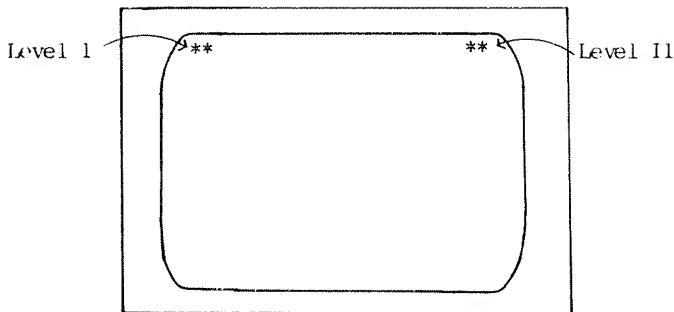
CTR-80 3.5 to 4.5 (user-generated); 4 to 5 (commercial tape)

Push **PLAY** key on cassette.

Type CLOAD or CLOAD "ℓ" and **ENTER** on Keyboard (where ℓ is letter identifying your program).

The cassette should be turning now.

Soon the TRS-80 video screen will show two asterisks.



If all is going properly, one asterisk will blink irregularly.

When it finishes loading, the cassette will halt the tape and the display screen will show

READY  
-

Depress the **STOP** button on the cassette (even though the tape has halted). *Do not leave the **PLAY** key depressed when not in use.*

Type LIST **ENTER**.

If the program seems OK, type RUN **ENTER**.

If no program is present (or if the asterisks appeared but did not wink) lower the volume slightly; rewind the tape and try again.

If something loaded, but the last part seems all jumble and gibberish, increase the volume slightly; rewind the tape and try again.



To SAVE a program currently in the computer

Set up the cassette recorder as described before; be sure plugs are set firmly in the proper holes.

Check the volume setting.

CTR 41        4-6  
                     (Try mid-range first.)  
 CTR 80        3.5-4.5

Rewind the tape. (Unplug REM jack only if on CTR-41 or use program fix.)

Reset the tape counter to 000.

Advance the tape to get past the "leader" if you are using standard tape—or to a blank place if the tape already contains some programs or data. (I usually begin programs at 10 on the tape and leave between 10 and 19 "counts" of blank tape between programs. This may be overly cautious, but it sure makes it a lot easier to find things.)

Note the program name and where it begins on the index card with your cassette or on the cassette label.

Depress both **RECORD** and **PLAY** keys on the cassette recorder (surprise—yes both should be down to record—so the TRS-80 can control things properly).

Type **CSAVE** " " (with a letter to identify the program between the " " ).

The tape should begin to advance and the TRS-80 will record your program on the tape.

When the program is recorded, the video screen will display as usual.

READY  
 >-

Depress the **STOP** button on your recorder. This is important.

Note where the program ended on your index card or the cassette label.

Rewind the tape to the beginning of the program (see index card and counter).

Depress **PLAY** only on cassette player.

Type **CLOAD ?** to check the tape against the program in the TRS-80 memory. If the tape checks properly, the screen will display **READY**, otherwise screen will display **BAD** and you should **CSAVE** " " it again.

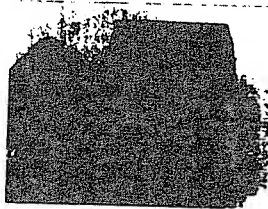
I record each program twice, just in case anything happens. You'll have more space than you can use on a 15-minute per side tape. (Use both sides, of course.)

### Sound and Music

Some TRS-80 programs include sound or music as part of their output. Higher quality musical reproduction is obtained by using a separate amplifier and speaker. If you are interested in medium fidelity sound rather than high fidelity music production, quite satisfactory results may be obtained using the amplifier present in the cassette player furnished with your TRS-80.

First get the program in your TRS-80 in the usual fashion. Then unplug the black plug from the ear hole in the cassette recorder (leaving the two grey plugs in place). Plug the earphone that came with the tape recorder into the ear hole (or better yet plug a speaker in there--you can get a suitable jackplug or converterplug from Radio Shack).

Now you need to fool the tape recorder. Remove any tape cassette from the recorder. Push in the little lever at the back left side of the opening where the cassette would go. While you are depressing it, depress both the **RECORD** and **PLAY** buttons on the front of the recorder. These should stay down when you release the lever. Now RUN your program.



The following instructions are used to place data onto tape and to retrieve data from tape:

PRINT #-1, PRINT #-1,X,Y,B\$,A\$

Output to cassette #1 from computer

INPUT #-1 INPUT #1,X,Z,B\$

Input from cassette #1 to computer.

If you are having trouble with keyboard bounce (double letters from a single key stroke), use the debounce tape. More recent keyboards seem not to have as much trouble with this as did those Level II's delivered before 1980.

### GAMES AND PLAYTOYS

Computers are magnificent game players. It isn't hard to play games which someone else has programmed - if they have done a good job of writing the program, but writing quality game-playing programs is difficult.

Here is a not very well written game. Put it on your computer; play it two or three times and then come back to the text and together we shall improve it.

```

5  REM      TREASURE HUNT - FIRST TRY
10  CLS
20  X1 = RND(50) : Y1 = RND(50)
30  PRINT "A TREASURE IS BURIED IN A 50 FOOT BY 50 FOOT SQUARE. IF YOU
    CAN LOCATE THE GRID POINT NEAREST IT, YOU WILL FIND THE TREASURE."
40  PRINT
50  INPUT "PLEASE TYPE COORDINATES X, Y = ";X,Y
60  IF X = X1 AND Y = Y1 THEN 500
70  A = X - X1 : B = Y - Y1
80  D = SQR(A*A+B*B)
90  PRINT "SORRY YOU MISSED IT BY";D;"FEET."
150 GOTO 40
500 REM      TO HERE ONLY IF TREASURE IS FOUND
510 FOR Q = 1 TO 3
520     PRINT @ RND(1000), "WHEEEE YOU GOT IT."
530 NEXT Q
540 GOTO 10

```

Well - it's a game - and the program works if you typed it in correctly. But it isn't much fun. You really need more information than just the distance to the treasure.

It would help if the program told you whether X and Y are too big or too small. Do you want to try to revise the program on your own before you continue?

One of the most common games computer programmers play is "one-up-man-ship". Modifying existing programs is one form of the sport. An important way to gain programming skill is to adapt and improve a working program written by someone else. There is an axiom among computer programmers that says

Any program you have written,  
I can improve upon.

It has as its converse

Any program I have written,  
you can improve upon.

and together they have a corollary

Any program I have written,  
I can improve upon.

Each of these is probably true. In some cases the real question is whether or not it is worthwhile to improve on a program even if you can do so.

The answer depends upon how frequently the program will be run, and how much of your time will be required to improve the program.

Some ways it is frequently possible to improve programs are:

1. Make them use less computer time.
2. Provide output better suited to the user's needs.
3. Make the program easier to understand (and hence easier to modify and maintain).
4. Generalize the program so it is useful in solving other problems without sacrificing speed.

Programs designed to play games or solve puzzles somehow seem particularly subject to improvement, even by beginners. We shall look at several commercially available programs and how they can be improved. This is one of the most satisfying ways to improve your own ability as a programmer.

Let's go back and improve the treasure hunt program. The "heart" of the original program is

```
.
.
.
50 INPUT "PLEASE TYPE COORDINATES X,Y =";X,Y
60 IF X = X1 AND Y = Y1 THEN 500
70 A = X - X1 : B = Y - Y1
80 D = SQR(A*A+B*B)
90 PRINT "SORRY YOU MISSED IT BY";D;"FEET."
150 GOTO 40
500 REM TO HERE ONLY IF TREASURE IS FOUND
.
.
.
540 GOTO 10
```

It would be nice to print another line of type right after 90 that tells us which way to go. Let's insert the following:

```
100 IF A < 0 THEN A$="TOO BIG" ELSE A$="TOO SMALL"
110 PRINT "YOUR X VALUE IS ";A$,
120 IF B > 0 THEN A$ = "TOO BIG" ELSE A$ = "TOO SMALL"
130 PRINT "YOUR Y VALUE IS ";A$
```

Insert these instructions and run the program again. It is more fun this way, isn't it? Play it several times.

HEY - DID YOU NOTICE THAT THE PROGRAM CHEATS? YES IT DOES.

If you get either X or Y correct, but not both correct, it tells you to change them both.

Insert temporary debugging instruction

```
21 PRINT X1, Y1
```

and run it again. The correct treasure location will now be printed at the top of the screen so you can test it. Try putting in two wrong values - Seems OK. Next put in the correct value for one coordinate and the wrong one for the other coordinate. Hmm - The program still claims both values are either TOO BIG or TOO SMALL. That won't do, will it?

OK                      OK                      OK

We'll fix that too. Try

```
100 IF A > 0 THEN A$ = "TOO BIG" ELSE IF A = 0 THEN A$ = "OK" ELSE
A$ = "TOO SMALL"
120 IF B > 0 THEN A$= "TOO BIG" ELSE IF B = 0 THEN A$ = "OK" ELSE
A$ = "TOO SMALL"
```

Try it - it works now. Remember to delete instruction 21 before you show it off to someone else.

There is really no reason we need always work on a 50 by 50 foot square. How about letting the player select the size of the square? We'll need an instruction to INPUT the size of the square.

```
15 INPUT "PLEASE TYPE LENGTH OF SIDE OF SEARCH SQUARE.";S
```

Remember to thwart the person who puts in 4.73 or -2 or some such joke.

```
16 S = INT(ABS(S))
```

Before continuing you should look through the program and see what other instructions need to be modified.

Instructions 20 and 30 need to be changed; our new program (still not perfect) is

```
5 REM      TREASURE HUNT - SECOND TRY
10 CLS
15 INPUT "PLEASE TYPE LENGTH OF SIDE OF SEARCH SQUARE.";S
16 S = INT(ABS(S))
20 X1 = RND(S) : Y1 = RND(S)
30 PRINT "A TREASURE IS BURIED IN A";S;" BY ";S;"FOOT SQUARE. IF YOU
  CAN LOCATE THE GRID POINT NEAREST IT, YOU WILL FIND THE TREASURE."
40 PRINT
50 INPUT "PLEASE TYPE COORDINATES X,Y =";X,Y
60 IF X = X1 AND Y = Y1 THEN 500
70 A = X - X1 : B = Y - Y1
80 D = SQR(A*A+B*B)
90 PRINT "SORRY YOU MISSED IT BY";D;"FEET."
100 IF A > 0 THEN A$ = "TOO BIG" ELSE IF A = 0 THEN A$ = "OK" ELSE
  A$ = "TOO SMALL"
110 PRINT "YOUR X VALUE IS ";A$
120 IF B > 0 THEN A$ = "TOO BIG" ELSE IF B = 0 THEN A$ = "OK" ELSE
  A$ = "TOO SMALL"
130 PRINT "YOUR Y VALUE IS ";A$
150 GOTO 40
500 REM      TO HERE ONLY IF TREASURE IS FOUND
510 FOR Q = 1 TO 80
520   PRINT @ RND (1000), "WHEEE YOU GOT IT."
530 NEXT Q
540 GOTO 10
```

Play the game a few times and then, after correcting the spacing on instructions 30, 100 and 120 if needed, show it to a friend.

What happens if someone inserts .5 or 0 as the value for S? Maybe we need another smart aleck guard at 17.

```
17 IF ABS(S) = 0 THEN S = 500 : PRINT "ALL RIGHT, SMARTY, I'LL FIX YOU.
TRY THIS."
```

Modifications of this treasure hunt program can provide the first approximation of a number of much more interesting games.

How about adding Depth at which the treasure is buried as well as X,Y locations?

Could you modify the coordinate game to play the child's game "BATTLESHIP" in which the computer selects positions for several targets which you must locate?

Can you insert graphics which will show you where your choices have been and the corresponding D value rather than telling you which way to move?

Try a few modifications of your own.

Here are some games programmed (or adapted) by secondary school students in Oklahoma in Spring, 1979. If you find unfamiliar instructions, see Lesson 16 or your TRS-80 LEVEL II BASIC Reference Manual.

```
5 CLS
7 PRINT "THIS IS A GUESSING PROGRAM. YOU ARE TO GUESS A FRACTION N/D
  WITH 1<=N<=10, 1<=D<=10."
9 PRINT
100 G = RND(10)/RND(10)
110 PRINT "I HAVE A NEW FRACTION FOR YOU TO GUESS. PLEASE TYPE YOUR
  GUESS IN DECIMAL FORM."
120 INPUT X
130 IF X = G THEN PRINT "WHEEE...YOU GOT IT!": GOTO 100
140 IF X > G THEN 170
150 PRINT "YOUR GUESS IS TOO SMALL. TRY AGAIN PLEASE."
160 GOTO 120
170 PRINT "YOUR GUESS IS TOO BIG. PLEASE TYPE IN ANOTHER GUESS."
180 GOTO 120
```

Play the game; then consider any desirable changes. One student suggested adding the following: *(However, it did not work as anticipated. Can you help?)*

```
10 ON ERROR GOTO 900
900 PRINT "PLEASE TYPE THE FRACTION USING DECIMAL FORM, NOT N/D FORM."
910 GOTO 9
```

THIS IS A HANGMAN GAME ADAPTED BY

```

3  REM      KAREN JEANNE HENRY
4  REM      R1 BX 580 MCLLOUD, OK 74851
5  REM      (405) 273-1133
10 REM      THE HANGMAN PROGRAM IS SELF EXPLAINED.  THE PERSON PLAYING
    WILL INPUT A LETTER AND THE COMPUTER WILL TELL YOU IF YOU
    ARE CORRECT OR NOT.
20 REM      AS YOU INPUT EACH LETTER THE COMPUTER MAY OR MAY NOT CHANGE
    GRAPHICS
25 REM      WHEN YOU FINISH THE GAME THE COMPUTER WILL VERIFY YOUR WIN OR
    LOSS BY CHANGING GRAPHICS AND/OR WORDS
30 REM      X + Y ARE USED IN SET STATEMENTS IN FOR-NEXT LOOPS
31 REM      D$( ) ARE THE WORDS, F$( ) ARE THE LETTERS OF THE WORDS
32 REM      WD IS THE TOTAL NUMBER OF WORDS IN THE MAIN PROGRAM
33 REM      W$ IS THE INPUTED LETTER L(N) + F1 ARE COUNTING DEVICES
34 REM      K IS A VARIABLE OF HOW MANY WRONG LETTERS
99  CLEAR 100
100 REM
102 CLS
103 WD = 10
104 K=0: FOR N=1 TO 8: L(N)=0: NEXT N
105 PRINT "***HANGMAN***"
106 FOR X = 9 TO 51 STEP 6
107 SET(X,15): SET(X + 1,15) : SET (X + 3,15) : SET(X + 2,15): NEXT X
108 FOR X = 70 TO 79: SET(X,44): NEXT X
109 FOR Y = 4 TO 44: SET(74,Y): SET(75,Y): NEXT Y
110 FOR X = 76 TO 101: SET(X,4): NEXT X
140 FOR Y = 5 TO 7: SET(101,Y): NEXT Y
150 IF FG = 1 THEN 160 ELSE D$(1) = "APPLE": D$(2) = "HEART": D$(3) =
    "STUDENT" : D$(4) = "TEACHER": D$(5) = "PENCIL" :D$(6) = "PAPER" :
    D$(7) = "PEN" : D$(8) = "WORK": D$(9) = "TIRED": D$(10) = "HARD"
160 DD = DD + 1: IF DD> WD THEN 2050 ELSE Z$ = D$(DD)
180 N = LEN(Z$)
190 F$(1) = MID$(Z$,1,1)
200 F$(2) = MID$(Z$,2,1)
210 F$(3) = MID$(Z$,3,1)
220 F$(4) = MID$(Z$,4,1)
230 F$(5) = MID$(Z$,5,1)
240 F$(6) = MID$(Z$,6,1)
250 F$(7) = MID$(Z$,7,1)
260 F$(8) = MID$(Z$,8,1)
270 IF F$(3)<>" " THEN 290
280 RESET (21,15) : RESET (22,15) : RESET (23,15) : RESET (24,15)
290 IF F$(4)<>" " THEN 310
300 RESET (27,15) : RESET (28,15) : RESET (29,15) : RESET (30,15)
310 IF F$(5)<>" " THEN 330

```



```

320 RESET (33,15) : RESET (34,15) : RESET (35,15) : RESET (36,15)
330 IF F$ (6) <> " " THEN 350
340 RESET (39,15) : RESET (40,15) : RESET (41,15) : RESET (42,15)
350 IF F$ (7) <> " " THEN 370
360 RESET (45,15) : RESET (46,15) : RESET (47,15) : RESET (48,15)
370 IF F$ (8) <> " " THEN 390
380 RESET (51,15) : RESET (52,15) : RESET (53,15) : RESET (54,15)
390 PRINT @ 388, "THIS WORD HAS "; N; "LETTERS",;
400 PRINT @ 452, "TYPE EACH LETTER" ;
410 PRINT @ 516, "PRESSING ENTER EACH TIME" ;
415 Z = 644
420 PRINT @ 896, " " ;
425 INPUT W$
427 PRINT @ 980, " " ;
430 FOR N = 1 TO 8
440 IF W$ = F$(N) THEN PRINT @ (258 +(3*N)), W$; : L(N) = 1 : FLAG = 1 :
    GOTO 450
450 NEXT N
460 IF FL = 0 THEN PRINT @ Z, W$; : Z = Z + 2
470 P = 0
480 FOR N = 1 TO 8 : IF L(N) = 1 THEN P = P + 1
500 NEXT N
510 IF P = LEN(W$) THEN 2000
520 IF FL = 1 THEN FL = 0 : GOTO 420
530 ON K + 1 GOTO 540, 660, 710, 810, 910, 980
540 K = 1
550 FOR X = 95 TO 107 : SET (X,8) : SET (X,16) : NEXT X
580 FOR Y = 10 TO 14 : SET (109,Y) : SET (93,Y) : NEXT Y
610 SET (94,9) : SET (108,9) : SET (108,15) : SET (94,15)
620 SET (97,10) : SET (105,10) : SET (101,11) : SET (101,12)
630 SET (95,12) : SET (96,13) : SET (97,14) : SET (98,14) : SET (99,14) :
    SET (100,14)
640 SET (101,14) : SET (102,14) : SET (103,14) : SET (104,14) :
    SET (105,14) : SET (106,13) : SET (107,12)
650 GOTO 420
660 K = 2
670 FOR Y = 17 TO 33 : SET (100,Y) : SET (101,Y) : SET (102,Y) : NEXT Y
700 GOTO 420
710 K = 3
720 FOR X = 103 TO 112 : SET (X,20) : NEXT X
750 Y = 19 : SET (117,14)
760 FOR X = 112 TO 116 : SET (X,Y) : SET (X+1,Y) : Y = Y-1 : NEXT X
800 GOTO 420
810 K = 4
820 FOR X = 90 TO 99 : SET (X,20) : NEXT X
850 Y = 26
860 FOR X = 85 TO 90 : SET (X,Y) : SET (X,Y-1) : Y = Y-1 : NEXT X
900 GOTO 420

```

```

910 K = 5 : Y = 33
920 FOR X = 103 TO 112 : SET (X,Y) : SET (X+1,Y) : Y = Y+1 : NEXT X
960 SET (113,43) : SET (114,43) : SET (115,43) : SET (116,43) :
    SET (117,43)
970 GOTO 420
980 Y = 42
990 FOR X = 89 TO 98 : SET (X,Y) : SET (X+1,Y) : Y = Y-1 : NEXT X
1030 SET (85,43) : SET (86,43) : SET (87,43) : SET (88,43) : SET (89,43)
1040 RESET (101,12) : RESET (95,12) : RESET (96,13) : RESET (106,13) :
    RESET (107,12)
1060 FOR X = 97 TO 104 : RESET (X,14) : NEXT X
1100 SET (97,14) : SET (98,13) : SET (99,13) : SET (100,13) :
    SET (101,13) : SET (102,13) : SET (103,13) : SET (104,13) :
    SET (105,14)
1110 PRINT @ 772, " AHHH! THAT'S TOO BAD"; : GOTO 2010
2000 PRINT @ 772, "CONGRATULATIONS! YOU WON!!!";
2010 FOR N = 1 TO 2000 : NEXT N : CLS : GOTO 100
2050 REM      THIS CHANGES WORDS
2055 ON F+1 GOTO 2060, 2070
2060 D$(1) = "GEOMETRY" : D$(2) = "ALGEBRA" : D$(3) = "TRIANGLE" :
    D$(4) = "FORMULA" : D$(5) = "SOLUTION" : D$(6) = "SUBTRACT" :
    D$(7) = "NEGATIVE" : D$(8) = "REDUCE" : D$(9) = "INVERSE" :
    D$(10) = "FRACTION" : K=1 : F=1 : GOTO 170
2065 DD=0 : FG=1 : GOTO 100
2070 REM      THE PERSON USING THIS PROGRAM CAN ADD MORE WORDS IN THIS SAME
    WAY, USING THE D$(SUBS), OR HE CAN USE DATA FILES AND STORE WORDS ON
    A TAPE RECORDER CONTROLLED BY THE COMPUTER. USE FOR N=1 TO 10 :
    INPUT #-1, D$(N) : NEXT N : GOTO 2065

```

This program will fit on a 4-K Level II TRS-80, but you may need to "squeeze out" most of the spaces as suggested in Lesson 11.

```
1060 FORX=97TO104:RESET(X,14):NEXTX
```

is hard to read, but requires less computer memory. Lines 3 to 34 may be omitted, if necessary.

After it is up and working, you may wish to try to improve it by including additional words from a cassette file. See Lesson 6.

R E V E R S E

```

10 REM      * REVERSE *          BOB YARBROUGH  4-78
20 REM      RANDOM DIGITS ARE FLASHED.  OPERATOR MUST ENTER NUMBER
21 REM      FORMED BY WRITING DIGITS IN REVERSE ORDER
30 REM      L IS INITIAL LENGTH OF STRING OF DIGITS
31 L=3
40 REM      A IS LENGTH OF COUNTER
41 A=1000
50 REM      G IS NUMBER OF CORRECT RESPONSES BEFORE STRING
51 REM      IS LENGTHENED
52 G=3
60 REM      STRING IS SHORTENED WHENEVER AN ERROR IS MADE.
70 CLS
80 PRINT "YOU MUST ENTER DIGITS SHOWN IN REVERSE ORDER."
90 PRINT
100 C=0
101 PRINT "NEXT STRING WILL BE ";L;" LONG."
102 Z=0
104 GOSUB 200
106 CLS
110 FOR X=57 TO 68
112     SET(X,21)
114     SET(X,29)
116 NEXT X
120 FOR Y=21 TO 29
122     SET(57,Y)
124     SET(68,Y)
126 NEXT Y
130 FOR N=1 TO L
132     A(N)=RND(9)
134     Z=10*Z + A(N)
136 NEXT N
140 FOR N=L TO 1 STEP -1
142     PRINT @ 542,A(N);
144     GOSUB 200
145     SET(60,25)
146     SET(61,25)
147 NEXT N
148 CLS
149 PRINT
150 INPUT P
152 IF P <> Z THEN 180
160 PRINT "CORRECT"
162 C=C+1
164 IF C < G THEN 102
166 L=L+1
168 GOTO 100

```

R E V E R S E (Continued)

```
180 PRINT "WRONG   ***';Z;''***'
183 GOSUB 200
186 L=L-1
190 GOTO 100
200 FOR M=1 TO A
202 NEXT M
204 RETURN
210 END
```

Note: If you wish the digits to be displayed for a shorter period of time, change line 41 to:

41 A = 500 or some smaller value.

ASTEROIDS

```

1  REM  *ASTEROIDS*   MICHAEL BRIGGS   1-79
2  CLS : PRINT " MANEUVER THE SHIP THROUGH THE ASTEROIDS BY USING THE
   UP-ARROW KEY TO GO UP, THE DOWN ARROW TO GO DOWN, AND THE < AND >
   KEYS FOR LEFT AND RIGHT."
3  Z = 15
4  DIMA(Z,15)
5  PRINT
6  INPUT "PRESS ENTER TO START" ; D
7  LL = 150
9  CLS
10 X = 63 : Y = 44
15  TI = 0
20  SET (X,Y) : SET (X,Y+1) : SET (X-1,Y+2) : SET (X+1,Y+2)
22  FOR G = 0 TO 127
24      SET (G,47)
26  NEXT
30  FOR L = 3 TO 15
40      FOR P = 1 TO Z
50          A(P,L) = RND(128)-1
60          SET(A(P,L),2*L)
70      NEXT : NEXT
80  PRINT @ 32, "GO!"
90  FOR L = 3 TO 13
100      R = RND(3)+2
110      FOR P = 1 TO Z
120          S$ = INKEY$
122          TI = TI + 1
123          IF S$ = "" THEN 240
125          RESET (X,Y) : RESET (X,Y+1) : RESET (X-1,Y+2) : RESET (X+1,Y+2)
130          IF S$ = "+" THEN Y = Y-1 : GOTO 170
140          IF S$ = "+" THEN Y = Y+1 : GOTO 170
150          IF S$ = "." THEN X = X+2 : GOTO 170
160          IF S$ = "," THEN X = X-2
170          IF X = 127 OR X = 1 GOTO 500
180          IF Y = 45 THEN GOSUB 1000
190          IF Y = 1 THEN 200
200          IF POINT(X,Y) OR POINT(X,Y+1) OR POINT(X-1,Y+2) OR
            POINT(X+1,Y+2) THEN 1505
235          SET (X,Y) : SET (X,Y+1) : SET (X-1,Y+2) : SET (X+1,Y+2)
240          RESET (A(P,L), 2*L)
250          A(P,L) = A(P,L)+R
260          IF A(P,L) >= 128 THEN A(P,L) = A(P,L) -128
265          IF ABS(A(P,L)-X) < 2 THEN 900
268          SET(A(P,L),2*L)
270          IF TI = LL THEN 2500
280  NEXT : NEXT
300 GOTO 90

```

*Note: NEXT works as well as NEXT X, but the practice of omitting the variable can cause a problem in nested FOR...NEXT loops.*

*Note: The + won't show on the screen, but it is in there.*

## ASTEROIDS (Continued)

```

500 CLS
510 PRINT "YOU HAVE JUST RUN INTO A BLACK HOLE, SORRY!"
520 GOTO 2020
900 IF 2*L >= Y AND 2*L <= Y+3 THEN 1505

910 GOTO 268
1000 CLS
1010 PRINT " YOU HAVE CRASH-LANDED, BUT IT'S NOT TOO FAR TO THE BASE."
1020 GOTO 2020
1505 FOR BL= 1 TO 40 : RESET (X,Y) : RESET (X,Y+1) : RESET (X-1,Y+2) :
      RESET (X+1,Y+2) : SET (X,Y) : SET (X,Y+1) : SET (X-1,Y+2) :
      SET (X+1,Y+2) : NEXT : CLS
1510 PRINT "(AP) HOUSTON - NASA MISSION CONTROL REPORTS THE LOSS OF AN
      INTER-PLANETARY SHUTTLE IN ROUTE TO SATURN. THE VESSEL APPARENTLY
      SUFFERED SERIOUS ASTEROID DAMAGE AND WAS UNABLE TO RETURN TO EARTH."
1520 GOTO 2020
2000 CLS
2010 PRINT "EXCELLENT NAVIGATION! CONGRATULATIONS ARE IN ORDER."
2015 PRINT "YOU STILL HAVE" ; (LL-TI)*10 ; "GALLONS OF FUEL LEFT."
2020 PRINT : INPUT "PRESS ENTER TO TRY AGAIN" ; D
2030 GOTO 9
2500 CLS : PRINT "YOU HAVE RUN OUT OF FUEL. YOU WILL CRASH-LAND IN
      APPROXIMATELY 15 MINUTES. WE MIGHT EVEN SEND A SEARCH PARTY TO
      PICK UP THE PIECES." : GOTO 2020

```

Here is a brief program that clutters up the screen in an interesting fashion. Try it. You might like it.

```

100 POKE 16396,23
110 CLS
120 FOR N=14336 TO 15360
130 PRINT CHR$(PEEK(N)) ; " "; Don't overlook the final ; here.
140 NEXT N
150 GOTO 100

```

Then type RUN **ENTER** and hit several keys at once without using **ENTER**. Try **Q** **X** **W** and then try various other keys with some or all of the above.

+	S
7	S
W	C

To terminate the program, open the little door on the back left-hand side of your keyboard unit and depress the button on the extreme left side. You may need to turn your computer off and then back on again to assure proper functioning of all keys.

You will find dozens of game programs in Softside, Computronics, Creative Computing, TRS-80 Computing (San Luis Rey, CA), PROG-80 (Milford, NH) and many other journals mentioned in Lesson 15. Some commercial games are rather expensive, costing \$25 and more. Such games are still cheaper than ones you design, write and debug, if your time is valuable. Other tapes are available at a cost so low that you can hardly afford the time to type in the program and correct your typing errors. People's Software will send you a list of available program tapes (1980 price \$7.50 for tape + 50¢ postage in U.S. for some tapes). They add more to the list all the time and will swap you free tapes for good original programs which you have written.

Tape 1 (\$8) contains a whole collection of games, some trivial, some excellent. Other programs deal with business, banking, bio-rhythms, speed-reading, math refresher, etc. Tape 2 (\$8) contains some slightly more sophisticated banking and business programs, math programs, statistical programs, etc. Tape 5 ( \$8 ) is more of the same. Tape 6 is People's Pascal, Version II, and the cost is \$23.50. You may or may not be ready for the Pascal language.

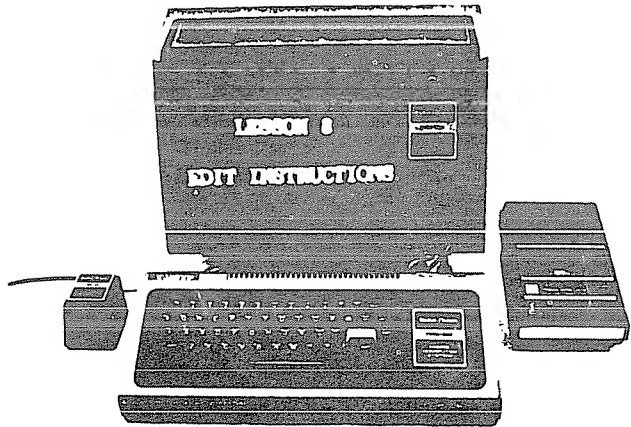
You can learn a lot by ordering one of these tapes and then examining, adapting and improving the programs to suit your whim. BE SURE TO SPECIFY WHETHER YOU HAVE LEVEL I OR LEVEL II BASIC. The address is:

People's Software  
 Computer Information Exchange  
 Box 158  
 San Luis Rey, CA 92066  
 Telephone: (714) 757-4849 (VISA or MASTERCHARGE accepted for phone orders)

PRACTICE SESSION 7

1. Improve the treasure hunt program introduced in this lesson.
2. Either devise a number guessing game of your own or improve the game written by a secondary school student in this lesson.
3. Play Karen Henry's modified Hangman program and then modify it to use your cassette tape recorder to provide additional words.
4. Either write a game-playing program of your own or improve a game-playing program written by someone else. Almost any game-playing program can be modified and improved. Try out your own ideas.
5. See if you can determine where the "fuel supply" is stored in the ASTEROIDS program. Change the fuel allotment. Also, determine whether fuel is used up by "time elapsed" or by "number of keystrokes" and plan your playing strategy to take advantage of this knowledge.
6. Modify Bob's program to play REVERSE so the player can INPUT a value to determine the amount of time the numbers are displayed (the difficulty level).
7. Look in one of the journals listed in Lesson 15 and find a computer game. Put the game on your computer, debug and play it. Then, see if you can improve it.





### EDIT INSTRUCTIONS

The simplest way to change short lines in a program is certainly just to retype it, as we have been doing.

If you mistyped a program as

```
1000 FOR K = 1 BO 17          (error)
1010 PPRINT K;              (error)
1020 NEXT K
```

RUN ENTER

The computer will display

```
? SN ERROR IN 1000
READY
1000_
```

By depressing key L the computer will list line 1000 and again display the line number, to facilitate correction.

```
?SN ERROR IN 1000
READY
1000 FOR K = 1 BO 17
1000_
```

You may now either space over to just under the error using the space bar, or you may depress S and the B (the character in error). In either case, you will now have

```
?SN ERROR IN 1000
READY
1000 FOR K =1 BO 17
1000 FOR K =1 _
```

Next, depress  for "change" and depress  followed by  to correct the line.

Type LIST  to see that the change was made.

```
1000 FOR K = 1 TO 17
1010 PPRINT K;
1020 NEXT K
```

If you type RUN  you can go through a similar error detection and error correction routine for line 1010, but if you notice the "PP" when you LIST it, it is easier just to retype it as

```
1010 PRINT K; 
```

If you didn't note the error and type RUN , you will obtain

```
RUN
?SN ERROR IN 1010
READY
1010 _
```

Upon depressing  the last line will show:

```
1010 PPRINT K;
1010 _
```

If you space the underline (cursor) over to directly under either of the letter P's

```
1010 PPRINT K;
1010 P_
```

and depress key , the computer will display:

```
1010 PPRINT K;
1010 P!P!
```

The !! indicates the portion that will be deleted when you depress .

Depress

Type LIST  and, sure enough, it shows:

```

1000 FOR K = 1 TO 17
1010 PRINT K;
1020 NEXT K

```

and RUN  produces the expected result.

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
READY
-

```

You can always go into the EDIT mode to change a line by typing EDIT *line number*. For example, to change the 17 in line 1000 to a 37, one merely types:

```
EDIT 1000 
```

The computer responds

```
1000 _
```

Depress , and you will have

```

>EDIT 1000
1000 FOR K = 1 TO 17
1000 _

```

Note that the "1" of 17 is the second "1" in line 1000, so depress

and then  and then  (Literally, "Seek the 2nd character 1".)

The computer displays

```

>EDIT 1000
1000 FOR K = 1 TO 17
1000 FOR K = 1 TO _

```

Now, depress  followed by  which gives

```
1000 FOR K = 1 TO 3_
```

Depressing  will complete the line and enter the changes into the program, as LIST  will then show, and RUN  will demonstrate.

Chapter 9 in your TRS-80 Level II BASIC Reference Manual gives additional details, which are summarized here.

# EDIT MODE SUBCOMMANDS AND FUNCTION KEYS

<u>Subcommand/Function Key</u>	<u>Function</u>
EDIT <i>line no.</i> <span>ENTER</span>	where <i>line no.</i> is a valid line number containing an instruction. This sends the computer to EDIT that line.
<span>ENTER</span>	End editing and return to Command Mode.
<span>SHIFT</span> <span>↑</span>	Escape from subcommand and remain in Edit Mode.
<i>n</i> <span>Space-Bar</span>	Move cursor <i>n</i> spaces to right.
<i>n</i> <span>←</span>	Move cursor <i>n</i> spaces to left.
<span>L</span>	List remainder of program line and return to beginning of line.
<span>X</span>	List remainder of program line, move cursor to end of line, and start Insert subcommand.
<span>I</span>	Insert the following sequence of characters at current cursor position; use <span>SHIFT</span> <span>↑</span> to get out of insert subcommand.
<span>A</span>	Cancel changes and return cursor to beginning of line.
<span>E</span>	End editing, save all changes and return to Command Mode. Same as <span>ENTER</span>
<span>Q</span>	End editing, cancel all changes made and return to Command Mode.
<span>H</span>	Delete remainder of line and insert following sequence of characters; use <span>SHIFT</span> <span>↑</span> to get out of this insert subcommand.
<i>n</i> <span>D</span>	Delete specified number of characters <i>n</i> beginning at current cursor position.
<i>n</i> <span>C</span>	Change (or replace) the specified number of characters <i>n</i> using the next <i>n</i> characters entered.
<i>n</i> <span>sc</span>	Move cursor to <i>n</i> th occurrence of character <i>c</i> , counting from current cursor position.



Delete all characters from current cursor position up to nth occurrence of character c , counting from current cursor position.

Let us try to use some of these EDIT instructions. Type along on your TRS-80 to help you follow what is taking place.

If you have the program line:

1000 PLEASE PRINT THIS LINE.

and type


RUN 

your screen will show

```

RUN
?SN ERROR IN 1000
READY
1000 --

```

If you depress  the screen will show

```

>RUN
?SN ERROR IN 1000
READY
1000 PLEASE PRINT THIS LINE.
1000 --

```



By now you realize that you need a PRINT instruction and that the message needs to be in quotes if it is to be printed.

Depress  key

(Nothing seems to happen, but it has put the editor into the insert mode.)

Type  
PRINT "

(which will appear on the screen)

Depress  

(Again nothing seems to happen, but this releases you from the insert mode.)

Depress 

(This moves to the end of the line and puts the editor back into insert mode.)

Your TRS-80 will now show

```
?SN ERROR IN 100
READY
100 PLEASE PRINT THIS LINE
100 PRINT 'PLEASE PRINT THIS LINE._
```

Note that the underscore at the end of the line shows you where the next character will appear. It is right where you need to insert another quote. To do so, type

"

Your correction is now complete.

If you type

RUN

the program will print the desired message.

Note: If, back at the beginning when you ran 100 PLEASE PRINT THIS LINE and your screen showed

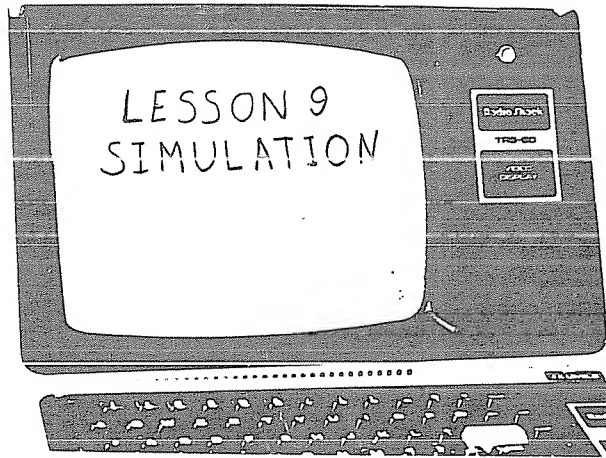
```
>RUN
?SN ERROR IN 100
READY
100 _
```

you forgot to give the EDITOR a command by depressing  (or some other command key) and just started to type PRINT, an unexpected action would have taken place.

The editor would ignore   as unacceptable commands. It would then use  as the "insert" command and would have inserted N T.

This is not at all what you wanted.

EDIT takes a bit of care, but it is well worth the effort.



### SIMULATION

A SIMULATION is a charade that uses a mathematical model to represent life-like experiments. The quality of the simulation is highly dependent upon how well the mathematical model represents the actual physical or social situation under study.

Many times it is difficult to compute a result of an actual experiment, either because the computation itself is too difficult to perform, or because the theory is not yet sufficiently developed. Even if the theory isn't well developed, it may still be feasible to use simulation to study the situation.

Simulation is one of the most important uses of modern computers. A simulation is a caricature. As in any caricature, it emphasizes the most prominent features under consideration and ignores the others.

We shall not attempt to simulate complex processes here. Instead, we shall simulate several rather simple situations — problems which you quite possibly could solve without resorting to simulation, but which will, nonetheless, illustrate the spirit of the game.

The heart of most simulation processes is some sort of random number generator.

The TRS-80 has an easy-to-use random number generator which you already investigated and used in Lesson 4.

- RND( $\emptyset$ ) produces a random decimal between 0 and 1, not including either endpoint.
- RND(K) produces a random integer between 1 and K, including both endpoints.
- RANDOM is sometimes used once at the very beginning of programs that will be used frequently. It reseeds the random number generator so that the same sequence of RND ( ) values is not produced on each run.

Let's begin with a simple program to simulate five rolls of two dice. Each die will produce one of the values 1, 2, 3, 4, 5, 6 at random. We wish to display the value shown on each die as well as their sum. For convenience, the individual dice are represented by R and G (for Red and Green, to tell them apart).

```

100 REM DICE THROWING PROGRAM
110 FOR K = 1 TO 5
120   R = RND(6)
130   G = RND(6)
140   S = R + G
150   PRINT "DICE SHOW" ; R ; G , "SUM =" ; S
190 NEXT K

```

Get this up and running before you go further.

Now imbed the above in a program that will collect statistics on how frequently each sum occurs in 100 rolls.

```

10 DIM T(12)
20 REM ZEROS T(2) TO T(12) FOR FUTURE RECORDS OF TOTALS
30 FOR S = 2 TO 12
40   T(S) = 0
50 NEXT S

100 REM DICE THROWING PROGRAM
110 FOR K = 1 TO 100
120   R = RND(6)
130   G = RND(6)
140   S = R + G
150   PRINT "DICE SHOW" ; R ; G , "SUM =" ; S
160   T(S) = T(S) + 1 : REM THIS ACCUMULATES TOTALS IN T( )
190 NEXT K

300 REM PRINTING OF TOTALS
310 PRINT
320 FOR S = 2 TO 12
330   PRINT S ; "WAS ROLLED" ; T(S) ; "TIMES IN 100 ROLLS OF TWO DICE."
340 NEXT S
350 PRINT

```

Once the program is running properly, you may change instruction 110 to  
 110 FOR K = 1 TO 1000  
 to obtain a larger sample size. You will wish to change 100 to 1000 in instruction 330 as well. Additional speed may be obtained by deleting instruction 150 once you are sure the program is functioning as desired.



### A BIOLOGICAL SIMULATION

#### Example 9-1

Suppose a biologist takes 100, equal-sized drops of liquid from a liter flask containing bacteria which are randomly distributed (i.e., no tendency to cluster). Upon testing, it is discovered that 50 of the samples are bacteria-free, while the other 50 contain 1 or more bacteria per sample.\* Assuming that the bacteria are distributed at random among the samples, approximately what is the total number of bacteria in the 100-drop sample?

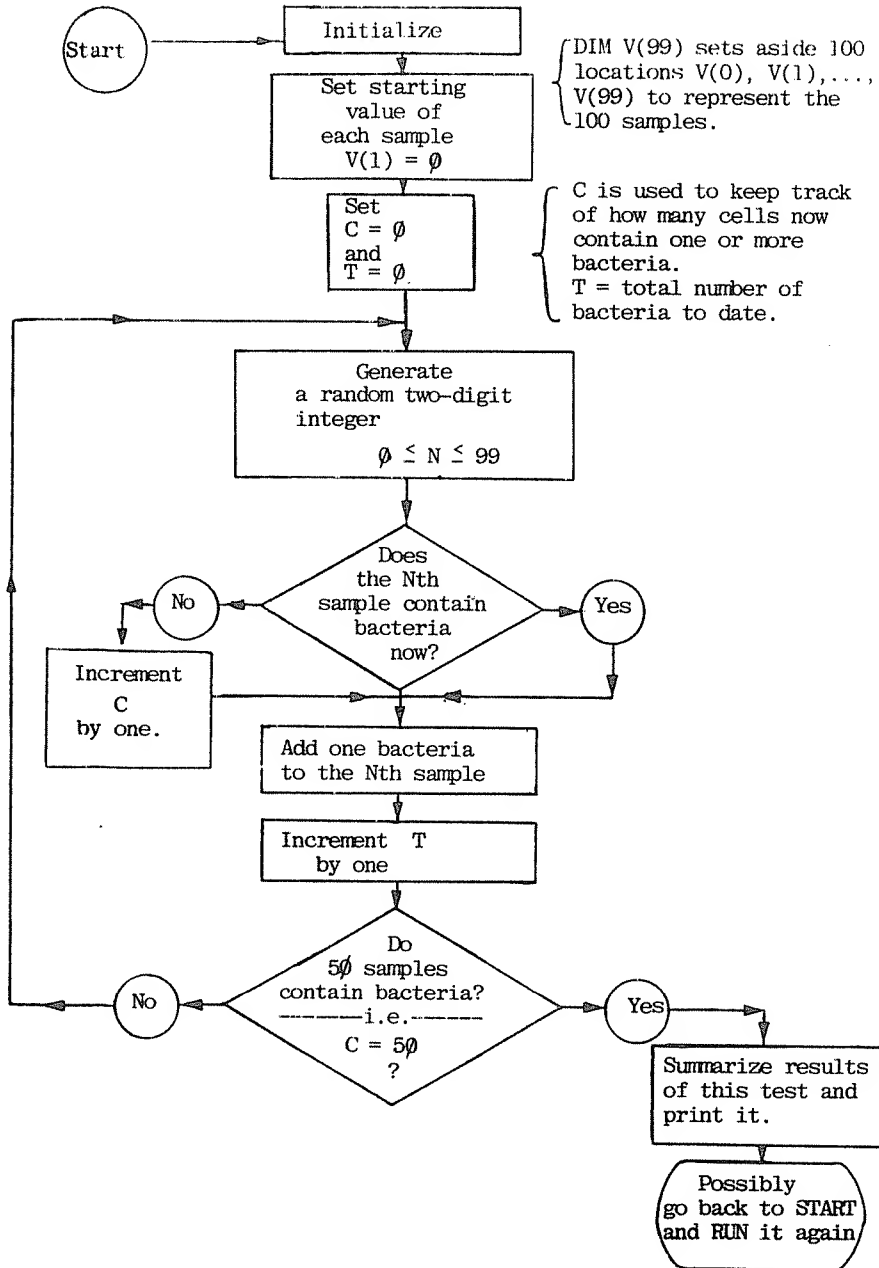
The exact answer cannot be determined. However, it is possible to obtain a statistical estimate of the desired number. You may or may not have sufficient mathematical sophistication to solve this problem directly, but it is an easy matter to simulate it.

Number 100 locations  $V(0)$  to  $V(99)$ , and store zero in each. See Lesson 10 for additional information on subscripted variables. Then generate a sequence of random two-digit numbers. Each time the number  $N$  is generated ( $0 \leq N \leq 99$ ) we add one to location  $V(N)$ . This is continued until exactly 50 of the locations contain non-zero numbers and 50 still contain zeros. At this stage, we have an estimate of the total number of bacteria in the 100 samples, either by having kept track, or by summing the  $V(N)$ 's.

Of course, the estimate may not be very good if we perform the experiment only once. By running it several times, using different random number sequences, we can obtain an average estimate that may be reasonably close. The "proper" number of runs is a mathematical problem of greater difficulty than we shall discuss here.

Do you see that this technique might be helpful? Actually, the mathematics needed to justify and analyze techniques of this type is highly sophisticated, but fortunately for many users, the techniques themselves are comparatively simple to use. A flow-chart of a computer program to simulate this experiment is given on the following page.

\* The testing consists of adding a drop of reagent to the sample drop. If it is bacteria-free, it will remain clear, but if the sample contains one or more bacteria, it turns purple. However, the sample does not turn twice as purple if two or more bacteria are present, so our test gives no idea of the total number of bacteria present.



```

100 DIM V(99)
110 FOR K = 0 TO 99
120   V(K) = 0
130 NEXT K
150 C = 0
160 T = 0

```

*C=Number of Cells containing bacteria.  
T=Total number of bacteria.*

```

200 REM GENERATE N: 0 <= N <= 99; INCREMENT V(N),
    TOTAL, AND IF APPROPRIATE COUNT.
210 N = RND(100)-1
220 IF V(N) > 0 THEN 230
225 C = C+1
230 V(N) = V(N) + 1
240 T=T+1
250 IF C < 50 THEN 210

```

```

300 PRINT "THE 100 DROPS CONTAIN A TOTAL OF" ; T ; "BACTERIA."

```

Before expanding this experiment, run it a few times. If it shows the same result each time, you may need to add the instruction

```

5 RANDOM or 5 RANDOMIZE

```

to prevent the program from resetting the random number generator to the same place at the start of each run. After the program is debugged and running, you could imbed it in a FOR...NEXT loop to run it, say, 100 times and give information on the 100 trials. You might even run it 1000 or 5000 times and compute mean and standard deviation of the total number of bacteria to be expected in the 100-drop sample. You might also consider how to change the program if the number of contaminated drops found in the 100-drop sample were 35 or 70 instead of 50.

For now merely add:

```

50 G = 0 : REM G WILL BE USED TO ACCUMULATE THE GRAND TOTAL OR
    CUMULATIVE TOTAL OF BACTERIA IN 100 RUNS

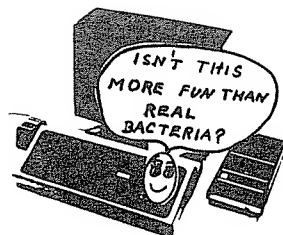
```

```

105 FOR S = 1 TO 100
300   PRINT "SAMPLE";S;"CONTAINS A TOTAL OF"; T ; "BACTERIA."
310   G = G + T
320 NEXT S
400 PRINT
410 PRINT " THE HUNDRED RUNS PRODUCED AN AVERAGE OF "; G/100 ; " TOTAL
    BACTERIA PER RUN "

```

In Practice Set 9 you will modify this program so that the number of runs can be specified by the user. You may wish to think a bit about which instruction will need to be changed. Actually, the simulated average of 100 runs should be expected to agree quite closely with the answer of 69 bacteria obtained using statistical theory.



## A SECOND BIOLOGICAL SIMULATION

### EXAMPLE 9-2

Consider a superficially related, but actually different example. This time our biologist has a liquid which he knows contains exactly 200 amoebas (or fish) which again are distributed at random throughout the liquid. If he separates the liquid into 50 equal size samples  $V(1)$ ,  $V(2)$ , ...,  $V(50)$ , how many samples will be amoeba-free? (Also, how many will contain 1, 2, 3, 4, 5, 6, more than 6 amoebas?)

The computation on this is well within your understanding, but that is not really the purpose of this example. Let's simulate the problem by creating 50 storage cells, each of which initially contains zero amoebas. This time generate exactly 200 pseudo-random numbers  $N$ , with  $1 \leq N \leq 50$ . Each time the number  $N$  appears, add 1 to the number in location  $V(N)$ .

100 DIM V(50), M(8)	<i>Sets aside space for samples V() and final data collection M().</i>
110 FOR K = 1 TO 50	<i>This zeros 50 samples V(K).</i>
120 V(K) = 0	
130 NEXT K	
200 FOR B = 1 TO 200	<i>This distributes the 200 amoebas into the 50 samples.</i>
210 N = RND(50)	
220 V(N) = V(N) + 1	
230 NEXT B	
300 REM ZERO THE M'S TO COLLECT DATA FOR FINAL REPORT	
310 FOR L = 0 TO 8	
320 M(L) = 0	
330 NEXT L	

*(Continued on next page)*

```

400 REM COLLECT DATA ON HOW MANY V-SAMPLES CONTAIN L AMOEBAS
410 FOR B = 1 TO 50
420   N = V(B)
425   IF N < 7 THEN 430
428   N = 8
430   M(N) = M(N) + 1
440 NEXT B

500 REM PRINT OUT SUMMARY
510 FOR L = 0 TO 6
520   PRINT " THERE ARE " ; M(L) ; " SAMPLES WITH EXACTLY " ; L ;
      " AMOEBAS "
530 NEXT L
540 PRINT " THERE ARE " ; M(8) ; " SAMPLES CONTAINING 7 OR MORE AMOEBAS "

```

Once you have the program up and debugged and feel it is running well, you may wish to include it in a FOR... NEXT loop to run the experiment several times and to tabulate the data in a more useful form. This is left for you to do. Once again it will be discovered that the average of, say 100, runs of such a simulation will agree quite closely with the forecast obtained using statistical theory, in which the number of amoeba-free samples is

$$50\left(1 - \frac{1}{200}\right)^{200} \text{ which is approximately } 50 \cdot \frac{1}{e} \approx 18.4$$

The results of our simulation program correspond very closely to the theoretical results.

A simulation of this type can also produce convincing evidence that certain phenomena are not random. Let us assume that in your area P persons died during D days of this year. If we assume that deaths occur at random, then we are placing P markers into D cells just as we were in the previous example, except P = 200, and D = 50. In this case, the agreement between the outcome of the simulation and actual data is not good. Indeed, further analysis shows that deaths are not usually distributed at random, but rather that a high "successive-day-dependence" seems to exist with alternate high and low death periods. This too can be simulated, but the model is more sophisticated.

### EXAMPLE 9-3

Now examine another simulation. This is essentially an Ehrenfest model with three molecules (see your physics or chemistry teacher). For simplicity it is posed as a "Guppy" problem.

Assume your aquarium contains 3 pregnant guppies. Because of their gravid condition you prefer not to net the guppies, but instead to place a "transfer cage" in the aquarium. At the start there are no guppies in the transfer cage and three in the aquarium. As time goes on the guppies swim into or out of the transfer cage at random. When all three are in

the transfer cage, the door is closed and no further exchange is possible. The guppies are then transferred to a maternity aquarium. The number of guppies in the transfer cage at various times might be

0+1+2+1+2+1+0+1+2+1+2+3 end.

If you were unusually lucky, it might even be

0+1+2+3 end,

The problem is to determine the average number of arrows in the chain before all three fish are in the transfer cage.

Again there exist statistical techniques (for example Markov Chain techniques) which could be used to determine the desired result, but we prefer to simulate the process.

Before examining the solution proposed by your author, at least try your hand at flow charting the heart of this problem.

Here is one possible model. Use a random number generator to determine whether the next guppy goes into or out of the cage: If  $\text{RND}(\emptyset) \leq .5$  put a fish into the transfer cage. If  $\text{RND}(\emptyset) > .5$  move out one, if possible. When all three fish are in the cage, stop the simulation.

```

10 REM      FIRST GUPPY MODEL SIMULATION
20 C = 0 : REM      C = # OF GUPPIES IN CAGE
30 A = 0 : REM      A = # OF ARROWS
90 PRINT "HISTORY=";
100 R = RND (0)
110 IF R > .5 THEN 200
120 REM      HERE IF FISH ENTERS CAGE R <= .5
130 A = A + 1
140 C = C + 1
150 PRINT C;
160 IF C = 3 THEN 500
180 GOTO 100
200 REM      HERE IF FISH DOES NOT ENTER CAGE R > .5
210 IF C = 0 THEN 100
220 A = A + 1
230 C = C - 1
240 PRINT C;
250 GOTO 100
500 REM      HERE IF ALL 3 GUPPIES ARE IN CAGE
510 PRINT "SUCCESS AT LAST"
520 PRINT A; "ARROWS OR 'SWIM THROUGH'S' TOOK PLACE"
530 PRINT

```

After debugging the above model you may enclose it in a program to run 25 times and give the average number of arrows for the 25 runs at the end of the history.

The preceding model may or may not represent this situation. Let us leave that for a chemist or physicist or guppy breeder to decide.

Before we show our model to an expert, it might be well to consider possible objections he could reasonably be expected to raise. Hmm--well, it does seem that the probability of a guppy leaving the cage should be greater when there are two guppies in the cage and only one outside than when the situation is reversed--doesn't it? Consider another possible model.

#### ANOTHER MODEL

You may easily discover a better simulation technique than the one suggested above. Be sure your next simulation takes into account that when the cage contains two guppies, then the next movement is twice as apt to be from the cage to the aquarium, as from the aquarium to the cage.

If you have trouble setting up your simulation model, here is an idea. Use three counters, one to represent each fish. If a given fish is in the cage, the corresponding counter will contain a 1, otherwise it will contain a zero. Two possible histories are:

(1) Fish #1 → 0 → 1 → 1 → 0 → 1 → 1 end

Fish #2 → 0 → 0 → 0 → 0 → 0 → 1 end

Fish #3 → 0 → 0 → 1 → 1 → 1 → 1 end

which could be represented more compactly as

000 → 100 → 101 → 001 → 101 → 111 end.

Another possible history could be

(2) Fish #1 0 → 0 → 1 → 1 end

Fish #2 0 → 1 → 1 → 1 end

Fish #3 0 → 0 → 0 → 1 end

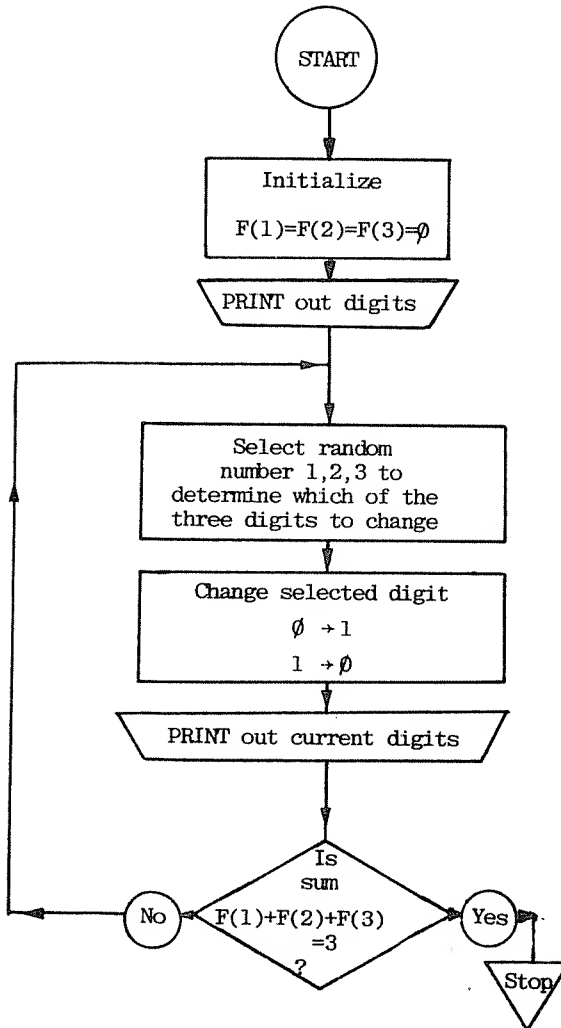
which could be represented as

000 → 010 → 110 → 111 end.

Before you continue, write a program or flowchart to simulate this model.

In this simulation all three fish start outside of the transfer cage, i.e.,  $F(1) = 0$ ,  $F(2) = 0$ ,  $F(3) = 0$ , THEN the computer generates a random number to determine which of the three guppies shall be the next one to change places (swim in if out, swim out if in cage). All three are in the cage when  $T = F(1) + F(2) + F(3) = 3$ .

The following flowchart summarizes the proposed program





Here is one possible program for the previous flowchart - but yours may be just as good, or even better than this one, so don't change yours - not yet, at least.

```

                                GUPPY - SECOND MODEL
10  REM GUPPY SECOND MODEL SIMULATION
20  DIM F(3)
50  REM SET F1 = F2 = F3 = 0 AS ALL GUPPIES ARE OUTSIDE CAGE
60  F(1) = 0
70  F(2) = 0
80  F(3) = 0
90  PRINT F(1) ; F(2) ; F(3) ; " -> " ;
100 R = RND(0)
110 IF R > .667 THEN 300
120 IF R > .333 THEN 200
130 REM HERE ONLY IF F1 IS TO MOVE
140 N = 1
150 GOTO 500
200 REM HERE ONLY IF F2 IS TO MOVE
210 N = 2
230 GOTO 500
300 REM HERE ONLY IF F3 IS TO MOVE
310 N = 3
500 REM NOW READY TO MOVE FISH #N
510 IF F(N) = 0 THEN 550
520 IF F(N) = 1 THEN 600
530 PRINT "SOMETHING IS WRONG. F(" ; N ; ") EQUALS"; F(N)
540 STOP
550 REM MOVE F(N) INTO CAGE
560 F(N) = 1
570 IF F(1) + F(2) + F(3) < 3 THEN 90
580 PRINT F(1) ; F(2) ; F(3) ; "END--ALL IN CAGE NOW."
590 PRINT : PRINT
595 GOTO 60
600 REM MOVE F(N) OUT OF CAGE
610 F(N) = 0
620 GOTO 90

```

Try it out. Debug the program and when it is running smoothly include it in a program to run the simulation 25 times and print out the average number of arrows for the 25 runs, at the end of the histories.

#### PRACTICE SET 9

1. Simulate the roll of a pair of dice. After your simulation is up and running, enclose it in an over-program that will roll the pair of dice 1000 times and keep track of how many times the sum shown is 2, 3, 4, ..., 12.
2. Compare the output of your simulation with the theoretical outcome, which you may either compute or discover in a book on probability, dice or gambling.

2. Modify the program of Problem 1 so it assumes dishonest dice have been used in which the 

·	·
---	---

 is replaced with a second 

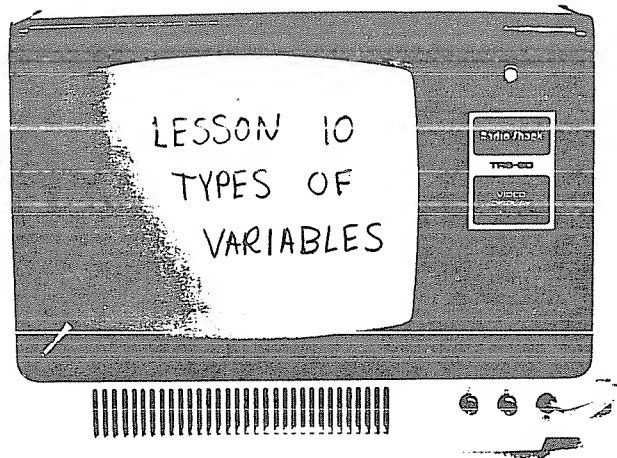
·	·
---	---

.
3. Modify the program of Example 9-1, so the number of runs can be specified by the user.
4. Enclose the program of Example 9-2 in a FOR...NEXT loop that will permit you to run the experiment 500 times and tabulate a summary of the data obtained.
5. Explain the purpose of instructions 425 and 428 in the program of Example 9-2. Did the given program ever use M(7)? Why do you think the author left it so?
6. Create a flow chart for your final revised version of Example 9-2 (Problem 4) and use it to explain what your program does to a friend.
7. A batch of 80 ounces of plastic contains 100 small metallic impurities. Assume that the plastic is well mixed and that the impurities are distributed at random throughout the plastic when it is molded into 160 half-ounce replacement heart valves. If a valve contains one or more impurities it must be discarded. It is quite possible that anywhere from 60 to 159 metal-free valves may be obtained from the given batch. Write a computer program to simulate one hundred 80-ounce runs and determine the average number of metal-free heart valves you would expect on the basis of your simulation.
8. Extend the program of Problem 7 slightly so that in addition to giving you the average of the hundred runs, it will also keep track of the largest and the smallest number of metal-free valves that were manufactured from a single batch on the hundred runs.
9. Assume that in Shire approximately 100 Hobbits die in a given 30-day period. Write a computer simulation that will print out a table of twenty lines of output data, each line representing a 30-day period with 100 random deaths distributed among the 30 days. Print a reasonably clear header before the 20 lines of output data, since the Shire Health Department wishes to use your data for comparison with actual death rates.
10. A well-known and interesting problem asks the question, "In a set of thirty persons selected at random, what is the probability that at least two of the thirty persons have the same birthday (anniversary date) (i.e., same month and date of birth, but not necessarily born in the same year)?" How would you simulate this problem?
11. Write a program to simulate Problem 10 assuming 365 days per year. After debugging your central program, expand it so you run your simulation for 50 sets of 30 persons before you print out the approximate simulated experimental probability. Compare your simulated result with that computed. (See "the birthday problem" in a text on probability, if you need help.) Open *Who's Who* to some page and take the first 30 birthdays you find for a similar experiment.

12. Simulate some simple game of your own choosing and have the computer summarize the results. Suitable games might be "matching pennies", "put and take", or some very simple dice game but not anything as complicated as "Yahtzee" or "craps".
13. Expand your program for Example 9-1 or Example 9-2 to run 1000 times and compute standard deviation as well as mean of the numbers involved.
14. Debug the first guppy model. After it is debugged and running, include it in an "over-program" that will run the model 25 times; printing a history each time and give the average number of arrows per run at the end of 25 runs. If your runs all produce the same history, there is apt to be something wrong with the way you invoked the random number generator. Fix it up and rerun the program.
15. Debug the second guppy model. After it is debugged (and checked that it doesn't always give the same history) incorporate your debugged program into a program that will run the model 25 times, print a history each time and give the average number of arrows per run. Compare the results of model 1 and model 2. Which do you feel is a better representation of the problem? Why?
16. Write a program to simulate a penny toss between two persons if the first person starts with P1 pennies and the second with P2 pennies, where P1 and P2 are input for the program. Your program should print out the number of pennies each person has after each toss and stop as soon as either person has lost all his pennies.
17. Write a program to simulate a ping-pong game with the initial input being two numbers (P1 and P2) which represent the probability of player 1 and player 2 returning a given shot successfully. Change your model a bit to make it more realistic if you can.
18. Same as Problem 17, but for the game of tennis.
19. If you were to write a simulation of a baseball game, what variables would you wish to be able to read in and thus vary from run to run? Discuss a possible simulation with a colleague. Make a rough flow chart of your proposed simulation.
20. The dice game known as "craps" is played by noting the total number of points, when two dice are rolled, as follows:
  1. If, on the first roll, you roll 7 or 11, you win.
  2. If, on the first roll, you roll 2, 3, or 12, you lose.
  3. If, on the first roll, you roll a 4, 5, 6, 8, 9, or 10, this number becomes your "point" and you continue rolling the dice until you either roll your "point" again or you roll a 7. If you roll your point again before you roll a 7, you win. If a total of 7 turns up first, you lose.

Use the computer to simulate a craps game. Have the output show, on one line, the actual history as well as the first win or loss of the game.

21. Rearrange your program of Problem 20 so that you can simulate the game of "craps" under the assumption that each die carries the spots 1-5-3-4-5-6, instead of the usual arrangement.
22. Write a program to simulate the deal of 52 playing cards with NC cards per player and NP players. Your program should read in the values for NC and NP and check that  $NC \cdot NP \leq 52$  before starting to deal. One way to deal using RANDU is to assume there are 52 cards numbered from 1 to 52 inclusive. You may wish to use DIMENSION C(52) and set the value of each C(I)=0 at the start, then once that "card I" is dealt, change the value of C(I) to +1 so that you do not deal the same card twice. (This causes hard feelings!) You may wish to store the hands in an array DEAL(NC,NP) by storing the numbers I, but in printing out the results, you should translate the stored integers into the usual suit and value designation. Use your program to deal 5-card poker hands to 8 players and also to deal 13-card bridge hands to four players.



#### VARIABLE TYPES

BASIC uses four distinct data types for its variables. Each variable may be forced to be any of the four types, either by using DEF statements or by the use of special type designators described below. The four types are:

Single Precision Floating-point (6 digits of accuracy)  
 Double Precision Floating-point (16 digits of accuracy)  
 Integer (between -32768 and +32767)  
 String (up to 255 alphabetic or numeric characters)

The first three types are used for numerical computation and storage. String variables are never used in arithmetic. They permit the storage, examination and comparison of sequences of letters, numbers, blanks and special characters.

##### a) Single Precision Floating-point Variables    DEFSNG A,B,W-Z

Single precision floating-point numbers are the type we have been using for most of our computation. It is the option which the TRS-80 automatically elects for you unless you instruct it to do differently. Such computation is carried out to (the binary equivalent of) 7 decimal digits of precision inside of the computer, but only 6 digits of precision are printed on the CRT screen.

Since single precision is the automatic default condition, it is not necessary to use DEFSNG unless you need to change the precision of variables previously defined as INTEGER or DOUBLE PRECISION.

##### b) Double Precision Variables

You are already aware (See Lesson 5.) that it is possible to

obtain variables of greater accuracy (16 digits instead of 6 digits) for a variable by inserting

```
DEFDBL X,Y
```

near the beginning of the program. This causes every variable beginning with the letter X or the letter Y to be double precision (17 digits internally, 16 digits displayed). It is also possible to indicate a range of initial letters in a DEFDBL statement. For example, DEFDBL A,C,S-W will cause every variable beginning with any of the variables A,C,S,T,U,V,W to be double precision unless it carries a special designator such as ! or % (see below).

#### c) Integer Variables

It is also possible to specify that some variables are of integer form rather than the usual floating-point numbers. On short programs it is not worth the effort to do so, but long-running programs frequently can be speeded up noticeably by inserting

```
DEFINT K,L,B-E
```

early in the program for those variables you know will always be integers in the range from -32768 to +32767. The DEFINT statement given above forces every variable beginning with K,L,B,C,D,E to be integer form. One of the most important savings is to make integers of the variables used in FOR...NEXT loops when appropriate. (Obviously, they must not take on fractional values in the step size.)

#### d) String Variables

It is also possible on your TRS-80 LEVEL II BASIC to define a string variable (indicated by annexing a \$ to any variable name) used to store strings (sequences) of letters and other characters up to 255 characters per string.

Examples A\$, C\$, GR\$, P1\$, P2\$

You may also define regular variables as string variables by using DEFSTR but be careful in doing this because if you designate DEFSTR A,D-G then every variable beginning with A,D,E,F,G will be a string variable and as such may not be used in arithmetic. Your authors usually prefer to use the annexed \$ to indicate a string variable.

Try this little program:

```
NEW 
10 CLS
100 INPUT "PLEASE TYPE YOUR NAME" ; N$
110 PRINT : PRINT : PRINT : PRINT
```

*continued on next page*

```

120 PRINT "HELLO", NS; "." : PRINT "I HOPE YOU ARE ENJOYING THIS COURSE."
130 GOTO 110

```

RUN ENTER

Try it out on your computer. Depress BREAK to stop the program.

The output is a bit fast to read isn't it? To slow down the program, make it count from 1 to 500 after it prints by inserting

```

125 FOR Q = 1 TO 500 : NEXT Q

```

RUN it again.

Try several different versions of your name.

Try inserting your social security number or phone number instead of your name.

String variables and string instruction are discussed further in Lesson 14.

#### e) Individual Variable Type Designators

It is also possible to make particular variables into integer variables by annexing a % sign after it — without changing other variables that begin with the same letter. For example

```

100 FOR X% = 1 TO 50
110   FOR K% = 1 TO 100
120     PRINT K%;
130   NEXT K%
140 NEXT X%

```

will run faster than

```

100 FOR X = 1 TO 50
110   FOR K = 1 TO 100
120     PRINT K;
130   NEXT K
140 NEXT X

```

The first program runs in about half the time of the second program

It is possible to have particular variables as regular single precision floating-point variables (6 digit precision) by annexing a ! to double precision floating-point variables (16 digit precision). Likewise, it is possible to have particular variables as double precision floating-point variables by annexing a # to any standard variable name. The type designator (% = integer, ! = single precision, # = double precision) takes precedence over an earlier DEF statement.

<u>Variable Type</u>	<u>Annexed Character</u>	<u>Examples</u>	<u>Typical Variables To Be Stored</u>
Integer Variable	%	A%, X%, K%, NI%	-2746, -5, 0, 1, 17, 3000
Single precision (6 digit) variable	!	A!, BX!, XI!	4, -50.73, .1276, 1.23456E+09
Double precision (16 digit) variable	#	A#, XI#, C#, BV#	1.234567887654321D+15

Note: If you wish to enter or use double precision constants, you can save a lot of trouble by placing a D at the end of it. Thus A# = 1234.7D. Also, use D in place of E to indicate the exponent of 10. Thus:

A = 6.02486E+23

A# = 6.024859316852145D+23

Note that B, B%, B!, B# and B\$ may all be used in the same program and each is a name for a different variable.

#### SUBSCRIPTED VARIABLES

The statement

DIM X(15), Y(15)

permits you to have sixteen different X-values

X(0), X(1), X(2), X(3), X(4), ..., X(14), X(15)

and sixteen different (associated?) Y-values

Y(0), Y(1), Y(2), ..., Y(14), Y(15)

in the computer at the same time.

Note that

X(3) and X3 are different variables, and each is different from X.

Such "subscripted variables" are important in statistics, where it is common to have sets of related data, say 25 temperatures and the 25 related product yields. The following program permits you to input 25 sets (T,Y) of temperature-yield data.

```

10  DIM T(25), Y(25)
100 FOR K = 1 TO 25
110     INPUT T(K), Y(K)
150 NEXT K

```

Note that we have not used location T(0) nor Y(0), but we could have.



The program might be expanded to also compute the average T value and the average Y value by accumulating the sum of the T's and the sum of the Y's. After the sums are computed, divide each by the number of data pairs (here 25) and store the result in T( $\emptyset$ ) and Y( $\emptyset$ )

```

10 DIM T(25), Y(25)
90 T = 0 : Y = 0
100 FOR K = 1 TO 25
110 INPUT T(K), Y(K)
120 T = T + T(K)
130 Y = Y + Y(K)
150 NEXT K
160 T(0) = T/25
170 Y(0) = Y/25

```

*(Note: This is only a fragment of a program. It permits you to enter data into T( ) and Y( ), but does not produce or PRINT anything.)*

Note that T and Y were used inside the FOR...NEXT loop rather than using T( $\emptyset$ ) and Y( $\emptyset$ ). This saves computer time. Most expert programmers consider it gauche to use a subscripted variable whose subscript does not change inside of a FOR...NEXT loop.

If desired, the program could be extended to compute standard deviation, correlation, mode, median, and other statistical measures. If you are familiar with statistics, you can extend the program yourself. You can even compute a "line of regression" or a "curve of regression" for the given data.

Our purpose was to introduce subscripted variables, a very handy tool for solving many problems.

### ARRAYS

It is also possible, in Level II BASIC to store arrays (matrices) of numbers (or letters or symbols). Consider a set of class grades for 20 students on each of six tests. Although it is possible to store student names, it is simpler to use student numbers instead.

The 20 by 7 array is:

	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
	Student ID no.	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
Row 1	12516	86	74	92	71	77	64
Row 2	12617	57	62	71	69	79	85
Row 3	77969	94	98	.	.	.	.

*continued on next page*

Row 4	38491	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
Row 20	21201	87	93	91	.	.	92

We represent it in the computer by

```
DIM G(20,6)
```

This gives an array with 21 rows and 7 columns, for storage. We shall not use row 0 for the storage data in this problem, but shall use column 0 to store the student's ID Number.

The 94 grade on test 1 of student number 3 (ID = 77969) is located in position (3,1). Position (2,4) contains the grade of 69 for the 2nd student (ID = 12617) on the 4th test.

The following program will read in the grades for the C-th test, to be supplied by the teacher. For C = 0, the student ID# may be typed in, if desired.

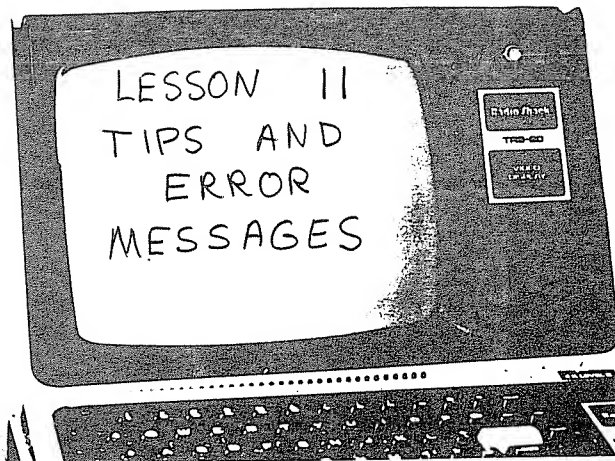
```
1  REM    PROGRAM TO INPUT QUIZ GRADES FOR CLASS OF 20 STUDENTS
10  DIM G(20,6)
100  INPUT "TYPE TEST NUMBER PLEASE"; C
110  FOR R = 1 TO 20
120    PRINT "TYPE GRADE ON TEST" ;C; "FOR STUDENT #"; G(R,0),
130    INPUT G(R,C)
140  NEXT R
```

The following program will display the entire array of scores, as well as computing and printing the average grade on the six tests for all 20 students. It assumes the data is already stored in the computer (from tape, or by hand, or ??). This program may be combined with that above. If no grades are entered, the computer will print zeros.

```
2  REM    PROGRAM TO DISPLAY AND AVERAGE STUDENT GRADES ALREADY IN THE
    COMPUTER
10  DIM G(20,6)
200  FOR R = 1 TO 20
210    T = 0
220    PRINT "STUDENT #" ;R, "I.D.#" ; G(R,0),
230    FOR C = 1 TO 6
240      PRINT G(R,C);
250      T = T + G(R,C)
260    NEXT C
270    PRINT "->"; T/6
280  NEXT R
```

Examine the program to understand how it works. If you have any questions, ask your instructor.

Note the use of indented instruction inside each FOR...NEXT loop. This is not important as far as the computer is concerned, but is a very helpful practice to help humans to perceive the structure of your programs. Your author sincerely recommends the practice.



We have collected a few tips and cautions that you may never need, but that you still may find helpful.

### TIPS

Don't let anyone spill beverage on or near your TRS-80. The only way to do this may be to forbid beverages near the computer. It is up to you, but a spilled Coke portends real grief forever afterward.

If you are plagued by double letters in your typing, it is because dust and dirt are forcing your TRS-80 to get multiple signals on a single key stroke. LEVEL II BASIC will have this problem more frequently than LEVEL I, since LEVEL II stores key strokes internally (great for fast typists). Keyboards manufactured in the 1980's seem to be less plagued with keyboard bounce than earlier TRS-80's. It is possible to remove the key tops and gently clean the contacts, but it is a lot easier to use the KEYBOARD DEBOUNCE (KBFIX) program furnished with LEVEL II TRS-80. It is a good idea to use a dust cover on the keyboard providing it is turned off (else heat may build up).

Begin your programs in line 100. Use lines 1 to 99 to identify yourself and program rather completely, as well as giving credit to any source you used before modifying your program.

If you would like your REM to stand out even more than it does by inserting 5 blanks after the REM, try depressing ☐ after typing REM and then type six or eight spaces before your REMark.

ERROR OS (out of string space) can frequently be remedied by inserting CLEAR n with a larger n value (say 100, 500, or MEM/4). The default value is CLEAR 50 whenever you turn on the TRS-80 or type NEW ☐.

If you use a BASIC program from a book or magazine, watch for RND(1) statements. Some BASIC's use RND(1) where the TRS-80 uses RND(0), namely, to generate a random decimal between 0 and 1.

If you type LLIST instead of LIST and do not have a line printer turned on, the TRS-80 will lock up. Open the small door at the back left of the keyboard unit and push the small round reset button.

The TRS-80 is designed for most room temperatures, but if you have clutter around it, that prevents air circulation. If you must use it in a hot place, put a couple of 1x1x5-inch blocks under the feet to permit extra air circulation.

Programs that produce a full screen of output can be ended with a tight loop (such as 9999 GOTO 9999) to prevent having break into the screen. Use **BREAK** to get out of a tight loop.

```
READY
>
-
```

If your program seems allright, but you get an SN ERROR that you can't identify, you may have depressed the **SHIFT** key while typing that line. (I tend to do this on **@**.) Retype the line in question, watching your **SHIFT** key carefully.

If you are an electronics buff, by all means get the TRS-80 Micro-computer Technical Reference Handbook (1980, \$9.95, Radio Shack). It has wiring diagrams you'll want to have. Remember, if you open the TRS-80 case, your guarantee is void.

Use CLOAD? to verify your Level II BASIC programs.

Double precision variables are not acceptable in FOR...NEXT loops.

If you have experienced problems with double precision variables not converting correctly (TRS-80 does not use decimal notation internally), it may be because you have not entered them correctly. If a double precision variable has fewer than 7 digits when typed, follow it with the letter D.

Run this to see what is meant.

```
100 S = .1
110 D# = .1
120 PRINT S,D#
```

Your output will look something like

```
.1      .1000000014901161
```

If you change 110 to

```
110 D# = .1D
```

all will be well. Try it and see.

Static electricity can play havoc with your TRS-80 and cassettes. If low humidity and carpet friction produce static electricity shocks in your computer room, do something about it. Remove the carpets or put down a non-static-producing rubber or plastic pad and increase the humidity in the room, if possible.

Don't force the 110V-plug on your video display into a socket or extension cord that is not designed to preserve polarity. One blade on the video 110V-plug is larger than the other--in this way, the video display chassis is grounded through the power cord. If you use an extension cord or multiple-outlet plug that does not preserve this polarity, you can get into trouble.

No doubt you have already discovered the HORIZONTAL HOLD and VERTICAL HOLD controls located on the rear of your video display. Once set, it should not be necessary to readjust them very often.

The BRIGHTNESS (B) and CONTRAST (C) controls are located on the front bottom right of your video display. Keep each in about the middle of the range. To lengthen tube life, DO NOT ADJUST TO MAXIMUM BRIGHTNESS OR CONTRAST. It may burn your tube if the display remains fixed.

Turn OFF the video display whenever you leave the computer, even if you leave the computer running.

Electrical interference from an elevator or other motor, a faulty fluorescent light or even the switch in a typewriter are not apt to interfere with your TRS-80 unless you have added disk to your system. If you are considering disk or other expansion interface items, you should add a couple of line filters at your power source as well.

SAVING MEMORY SPACE

If you have only 4K of memory, you may find that unwelcome OM ERROR message signifying all of the available memory has been used or reserved. There are several things you can do to conserve memory space.

Cut down on the size of any arrays used.

Use multiple statements per program line with a colon between statements.

Use integer variables whenever possible—in FOR K% = 3 TO 300, for example. Use the same variable in all FOR...NEXT loops that are not nested.

Delete unnecessary parentheses in arithmetic statements.

Cut down on both the number and length of string variables.

Delete unnecessary spaces in all statements.

Eliminate extra variables, reuse variables no longer needed, keep variable names short.

If necessary, also delete REM statements.

ERROR OM can sometimes be cured providing you are using no string variables by inserting CLEAR 0 at the beginning of your program.

OS ERROR frequently seems to be the result of a missing CLEAR n statement, where n is greater than 50. Try CLEAR 200 or or CLEAR MEM/4.

SPEEDING UP PROGRAMS

Although computers are fast, they do run into time difficulties on long programs. Usually the solution is to think hard about the algorithm being used. It is frequently possible to devise a much faster algorithm once the problem is fully understood. There are also programming techniques, known to most experienced computists, that save considerable time.

Don't use subscripted variables inside of a FOR...NEXT loop unless the subscript is changing.

Combine several statements per line (separated by colons).

Use integer variables in FOR...NEXT loops that have integer step sizes.

Define the most commonly used variables first--this puts them near the top of the variable table and saves look-up time if many variables are involved.

Don't recompute values needed repeatedly; store them instead.  
(This is particularly vital inside of nested FOR...NEXT loops.)

Use small data sets and short FOR...NEXT loops in debugging long programs.

Use faster forms whenever possible.

$B + B$  is faster than  $2 * B$ .

$.2 * C$  is faster than  $C/5$ . (Avoid division when possible.)

$D * D * D * D$  is much faster than  $D + 4$ .

Don't recompute trigonometric or log functions for the same value.

To evaluate  $Y = 5 \sin^4 X + 8 \sin^3 X - 2 \sin^2 X$

use  $S = \sin(X)$

$Y = ((5 * S + 8) * S - 2) * S * S$ .

Use POKE graphics. This can cut graphics time to 1/5 of SET(X,Y).

Use stored constants (a letter like B or P) rather than numeric constants in the statements that are executed frequently.

Probably the most important thing you can do to make programs run faster is to analyze which subpart of your program uses the greatest amount of time and then rewrite that portion as a subroutine in Z-80 machine language. Call the subroutine using the USR(X) routine. However, that requires learning another language. See Chapter 8 of your TRS-80 Level II BASIC Reference Manual under USR(X) and a text on Z-80 machine language. Radio Shack has a book, TRS-80 Assembly Language Programming by Wm. Barden, Jr., (Cat. #62-2006) which will be helpful if you are sufficiently interested to wish to learn Z-80 chip language.



ERROR MESSAGES

We have discussed, under EDITING, Lesson 8, the appearance on the screen of the message

?SN ERROR IN 1000

LEVEL II BASIC also has other messages that help you discover errors in a program. The following ERROR messages are among those you may encounter. (If you have a disk on your system—consult the appropriate reference manual for additional messages.)

EXPLANATION OF ERROR MESSAGES

- BS Subscript out of Range: An attempt was made to assign a subscripted variable or a matrix element with a subscript beyond the DIMensioned range.
- CN Can't Continue: A CONT was ENTERed at a point where no continuable program exists. e.g., after program was ENDED or EDITed.
- DD Redimensioned Array: An attempt was made to DIMension a matrix which had previously been dimensioned by DIM or by default statements. It is a good idea to put all dimension statements at the beginning of a program.
- FC Illegal Function Call: An attempt was made to execute an operation using an illegal parameter. Examples: square root of a negative argument, negative matrix dimension, negative or zero LOG arguments, SET (X,Y) with one value out of bounds, etc. Or USR call without first POKEing the entry point.
- FD Bad File Data: Data input from an external source (i.e., tape) was not correct or was in improper sequence, etc.
- ID Illegal Direct: The use of INPUT as a direct command, without a statement number.
- L3 DISK BASIC only: An attempt was made to use a statement or function which is available only when the TRS-80 Mini Disk is connected via the Expansion Interface.
- LS String Too Long: A string variable was assigned a string value which exceeded 255 characters in length. Break it into two or more strings.
- MO Missing Operand: An operation was attempted without providing one of the required operands.
- NF NEXT without FOR: NEXT is used without a matching FOR statement. This error may also occur if NEXT *variable* statements are reversed in a nested loop.

- NR No RESUME. End of program reached in error-trapping mode.
- OD Out of Data. A READ or INPUT # statement was executed with insufficient data available. DATA statement may have been left out or all data may have been read from tape or DATA.
- OM Out of Memory. All available memory has been used or reserved. This may occur with very large matrix dimensions, nested branches such as GOTO, GOSUB, and FOR...NEXT loops, or with large strings. Usually the program can be rewritten so that it uses less memory. See separate suggestions.
- OS Out of String Space. The amount of string space allocated was exceeded. Usually this can be fixed by starting program with CLEAR *n* for some *n* > 50 or by using CLEAR MEM/2.
- OV Overflow. A value input or derived is too large or small for the computer to handle.
- RG RETURN without GOSUB. A RETURN statement was encountered before a matching GOSUB was executed.
- RW RESUME without ERROR. A RESUME was encountered before ON ERROR GOTO was executed.
- SN Syntax Error. This usually is the result of incorrect punctuation, open parenthesis, an illegal character or a misspelled command.
- ST String Formula Too Complex. A string operation was too complex to handle. Break up operation into shorter steps.
- TM Type Mismatch. An attempt was made to assign a non-string variable to a string or vice-versa.
- UE Unprintable Error. An attempt was made to generate an error using an ERROR statement with an invalid code.
- UL Undefined Line. An attempt was made to refer or branch to a nonexistent line.
- /Ø Division by Zero. An attempt was made to use a value of zero in the denominator.

USING KEYBOARD COMMANDS IN YOUR PROGRAMS

TRS-80 Level II BASIC has an unusual extra "goodie" you may find handy. Commands such as RUN, NEW, LIST, DELETE 100-499, and similar keyboard commands can be inserted into and executed from a program. Try the following programs to get the feel of the idea.

```

100 CLS
110 FOR K=1 TO 30
120   PRINT K;
130 NEXT K
140 PRINT
RUN

```

Then add

```

150 RUN

```

and RUN it again.

Now, change 150 to

```

150 LIST

```

and RUN it again.

Depress BREAK and change 150 to

```

150 DELETE 100

```

LIST the program. RUN it, then LIST it again. Note that 100 has disappeared. Change 150 to 150 NEW.

and RUN it again.

It seems rather straightforward until you discovered that if you re-enter the program 100 to 140 and add

```

150 LIST
160 RUN

```

the program never gets to 160. Most commands (RUN is an exception.) send the computer back to READY mode instead of continuing the program. You can overcome this by using

```

PRINT @ 970, "PLEASE TYPE  RUN (ENTER)."

```

just before the DELETE or LIST command.

Try the program on the following page.

```

100 CLS
110 FOR K=1 TO 30
120   PRINT K;
130 NEXT K
140 PRINT : PRINT
150 PRINT "PLEASE TYPE   RUN (ENTER)"
160 DELETE 100-160
165 PRINT : PRINT : PRINT
170 PRINT "WHEN THIS IS LISTED NOW, LINES 100 TO 160 WILL BE MISSING."
180 PRINT : PRINT
190 LIST
RUN ENTER

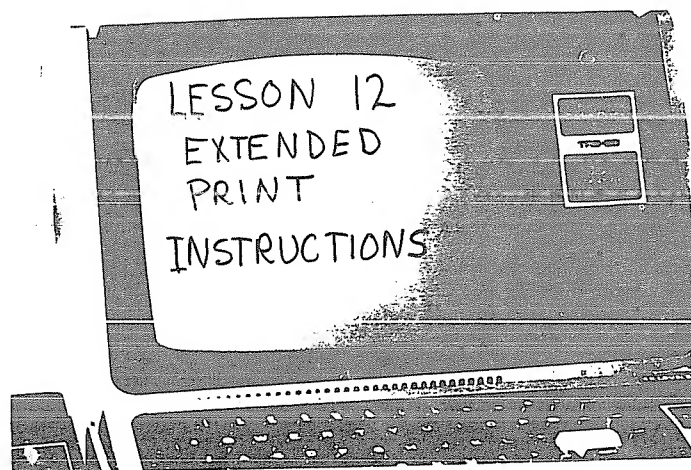
```

Advanced programmers use this technique to get extra space when OS ERRORS plague them. Early parts of the program can be deleted before the string variables are used. A sample might be:

```

10
.
.
.
100
.
.
.
499
504 REM   TRANSITION TO GET MORE STRING SPACE
505 CLS
510 PRINT : PRINT
515 PRINT "PLEASE TYPE   RUN(ENTER)"
520 DELETE 10-499
600 REM   MAIN PROGRAM USING STRINGS STARTS HERE
610 CLEAR 5000 : DIM B$(40), M$(50)
620 PRINT "THIS IS THE MAIN STRING PROGRAM."
700 LIST

```



You are already familiar with the instruction

```
PRINT
```

which gives a blank line and

```
PRINT A,B,C,D
```

which gives wide (4 column) spacing where a comma is used and close-packed spacing where a semicolon is used.

```
PRINT A; B,C,D
```

is also familiar (see Lesson 2).

BASIC has several other useful instructions including

```
PRINT TAB( )
PRINT USING A$,K
PRINT STRING$(K,"#")
```

EXTENDED PRINT INSTRUCTIONS

PRINT TAB (*expression*)

The "*expression*" may be a constant or a variable or a computed expression which is an integer between 0 and 255. This "tabs" the cursor to the indicated position. It is not possible to move the cursor to the left by tabbing. If the cursor is already to the right of the TAB expression, the TAB is ignored. Any indicated printing occurs at the place where the cursor is.

Try the following:

100 A = 3

110 B = 5

120 PRINT TAB(A) A; TAB(B) B; TAB(A+B) "A+B"; TAB(35) "35";  
TAB(10\*B+A) "Z"

on your TRS-80.

The TAB( ) function is particularly useful in graphing and the creation of special tables and forms.

PRINT USING string;value

It is possible to format the output in a more precise manner than is available on the ordinary PRINT instruction employing ; , and " " or even the PRINT @ instruction.

The PRINT USING instruction makes this possible. Try the following program.

```

10 CLS
900 PRINT "PLEASE TYPE YOUR NAME, INCLUDING MIDDLE INITIAL"
910 INPUT N$
1000 FOR K = 917 TO 1130 STEP 13.25
1010 PRINT "PAY TO THE ORDER OF "; N$; " ";
1020 PRINT USING "**$###.## DOLLARS"; K
1050 NEXT K

```

In instruction 1020, the "\*\*\$###.## DOLLARS" gives the format in which the material is to be printed. The \*\*\$ says to fill up any space before the first digit with asterisks and then a \$. The decimal point followed by two #-signs ( .##) indicates exactly two places beyond the decimal point are to be printed. Note the change in number of \* where the amount changes from \$996.50 to \$1009.75 in the output when you run this program. Consult your TRS-80 Level II BASIC Reference Manual, Chapter 3 for additional information.

PRINT USING statements may use any of the following field specifiers:

<u>Numeric Character</u>	<u>Function</u>	<u>Example</u>
#	Numeric field (one digit per #).	###
.	Decimal point position.	##.###
+	Print leading or trailing sign (plus for positive numbers, minus for negative numbers).	+#.### #.###+ -#.### #.###-
--	Print trailing sign only if value printed is negative.	###.##-
**	Fill leading blanks with asterisk.	**###.##

(Continued next page)

<u>Numeric Character</u>	<u>Function</u>	<u>Example</u> (continued)
\$\$	Place dollar sign immediately to left of leading digit.	\$\$#####.
**\$	Dollar sign to left of leading digit and fill leading blanks with asterisks.	**\$#####.
↑↑↑↑	Exponential format, with one significant digit to left of decimal.	#.#####↑↑↑↑
<u>String Character</u>	<u>Function</u>	<u>Example</u>
!	Single Character	!
%spaces%	String with length equal to 2 plus number of spaces between % symbols.	%      %



```
PRINT STRING$(K,"CHARACTER" or NUMBER)
```

This is very useful in graphing since it returns a string of K characters each of which is the character enclosed in quotation marks. It is also possible to use the ASCII number associated with a given character. The value K must be between 0 and 255, as must any number used for ASCII characters. See Lesson 4 for list of ASCII equivalents.

```
PRINT STRING$(25,"#") produces
```

```
##### (25 of them)
```

on your screen.

The following program plots the number K followed by a string of  $K^2$  symbols:

```
100 CLS
110 FOR K = 1 TO 10
120   L = K*K
130   PRINT K; STRING$(L,"%")
135   PRINT
140 NEXT K
150 GOTO 150
```

Try it on your screen and see what happens. Then change the symbol inside the " " to some other symbol of your choice.

Lesson 4 included a program to generate a bar graph (histogram).

```
5 CLEAR 200
10 CLS
100 PRINT @ 6, "GRAPH OF ENROLLMENTS AT THE UNIVERSITY OF OKLAHOMA"
110 READ Y,D
115 IF Y < 0 THEN 115
120 DATA 1940,21,1950,28,1960,32,1970,51,1980,59,-1,-1
130 PRINT Y; STRING$(D,143)
140 GOTO 110
```

The instruction

```
5 CLEAR 200
```

clears all variables and sets aside 200 bytes for string variable use. This is maybe more than we need.

110 READ Y,D reads two values from the DATA string.

Instruction

130 PRINT Y; STRING\$(D,143) prints the current value of Y (read in 110 from DATA) and follows this with a D-long string of the symbol having ASCII equivalent of 143 which is



The same bar graph results as in the Lesson 4 program using SET (X,Y): SET (X,Y+1), but in much less time and with a shorter program. The DATA statement is also easy to change in the new program.

Note that if your D value is greater than 59, the bar will be continued on a second line. However, D may be as great as 256 if desired, but you'll need to use CLEAR 256 if D is that large.

Here is another program that may interest you. Try it out yourself.

```
100 CLEAR 63
110 CLS
120 FOR Y = 0 TO 96 STEP 64
130   PRINT @ Y, STRING$(63,191)
140 NEXT Y
150 GOTO 150
```

After you see what it does, run the same program with 100 changed to

```
100 CLEAR 60      {Why did you get OS ERROR, if you did?}
```

You may also wish to change to 100 CLEAR 100 and then change STRING\$(63,191) to STRING\$(64,191). See if you can explain the unexpected dark band at the bottom of the screen when you RUN it.

Sometimes it is handy to be able to clear only a portion of the screen. Try this:

```
100 CLS
110 FOR K=1 TO 205
120   PRINT K;
130 NEXT K
140 PRINT
200 PRINT @ 256, CHR$(30); "THIS IS THE NEW LINE FROM STMT 200.";
300 PRINT @ 512, CHR$(30); "THIS CAME FROM STATEMENT 300.";
500 GOTO 500
RUN
```

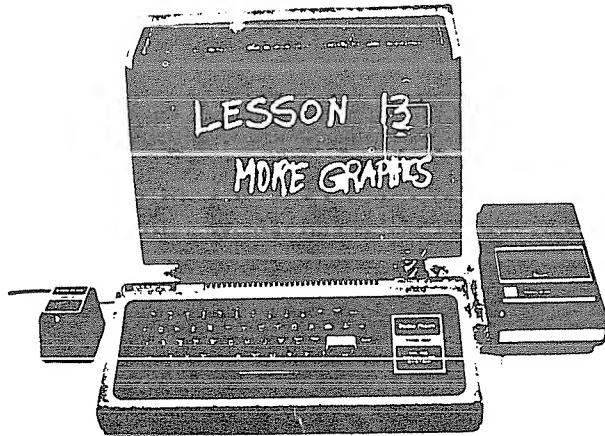
Depress BREAK to regain control.

After you have RUN this program a couple of time with the semi-colon at the ends of lines 200 and 300, remove one of the semicolons and RUN the revised program.

Investigate what happens if you insert:

```
299 PRINT CHR$(23);
```

You may also wish to use CHR\$(31) in statement 300 to see what happens.



### More Graphics

Lesson 4 provided an introduction to graphics that may be adequate for your needs. However, you should know that the TRS-80 has many graphic abilities in addition to the

CLS	clears screen
SET (X,Y)	turns on rectangular spot at (X,Y)
RESET (X,Y)	turns off rectangular spot at (X,Y)
POINT (X,Y)	tests to see if point (X,Y) is turned on, if so it returns a logical TRUE (-1), otherwise a logical FALSE (0). Can be used in IF instructions (See Lesson 14, for details.) on logical operators.
CHR\$(N)	produces action, letter or symbol that corresponds to code N where $0 \leq N \leq 255$

For example: the operator `ASC(symbol)` is the inverse of `CHR$(N)`. It produces the numeric (ASCII) code corresponding to the symbol in parentheses (the first symbol in the string, if the parentheses contain a string of more than one symbol). This is used in an excellent cipher program considered in Lesson 14 of this text.

In Lesson 4 you ran the following program:

```

10 CLS
100 FOR K = 33 TO 191
110   PRINT K,
120   FOR L = 1 TO 45
130     PRINT CHR$(K);
140   NEXT L
150   PRINT
170 NEXT K
200 GOTO 100

```

If your TRS-80 shows both upper and lower case letters (a modification you may obtain, if needed) `CHR$(K)` for  $K=65$  to  $90$  will display upper case letters A to Z while  $K=96$  to  $127$  displays the corresponding lower case (SHIFT) symbols corresponding to @, A to Z, +, +, +, +, —. Most TRS-80's display only upper case letters for both ranges.

Run it again now. Change instruction ~~100~~ to

```
100 FOR K = 20 TO 255
```

and run it again, after turning to the table given in Lesson 4 that discusses the effect of CHR\$(N) for various N values.

The pixel-blocks that appear corresponding to

```
PRINT CHR$(K)
```

for K between 129 and 191 are particularly useful in speeding up graphics. At first glance they may seem haphazard, but actually the choice of numbers associated with the graphic you wish to light up is both logical and easy to use, once it is explained. Each large block (letter & space between lines) is divided into six small rectangles (pixels)



The individual pixels are numbered.

1	2
4	8
16	32

To light up the pixels numbered 1,2,8 and 16,



add the sum of pixels used:  $1 + 2 + 8 + 16 = 27$

and add that sum to 128 giving

$$128 + (1 + 2 + 8 + 16) = 155.$$

The instruction

```
PRINT CHR$(155)
```

will light up the



desired pixels much faster than

will SET (X,Y).

Much time can be saved in graphics by using `CHR$( )`.  
You may use the instruction

`POKE address, pixel number`

where *address* is 15360 plus the number between 0 and 1023 which locates the position (see Lesson Four ) of the 6-block under consideration. The *pixel number* is the number (128 + (sum of desired pixels)).

For example:

The entire 6-block

1	2
4	8
16	32

has sum 63 so  $128 + 63 = 191$  is

the pixel number to turn on the entire 6-block.

The program

```
100 CLS
110 FOR X = 15360 TO 16383
120   POKE X, 191
130 NEXT X
140 GOTO 140
```

(See Lesson 16 for POKE instruction.)

RUN

will "white out" the entire screen in record time. It can be combined with:

```
140 X = 2 + RND(121) : Y = 1 + RND(43)
150 RESET (X,Y) : RESET(125-X,Y)
160 RESET (X,46-Y) : RESET(125-X,46-Y)
170 GOTO 140
```

to produce a reverse art pattern, if desired.

Try some additional experiments with the above 100-140 program such as combining it with

```
140 PRINT @ RND(1000), "HI your name";
150 IF RND (0) > .01 THEN 140 ELSE 100
```

or some other experiments of your own design. Remember as long as you keep beverages away from the TRS-80 and do not abuse it, you probably will not harm it by your programming experiments — and the best way to learn computing is to compute and then to think about the results before continuing.

If you need to insert a variable number of blanks in a line

```
PRINT CHR$(192+k)
```

will insert k blanks for  $k = 0, 1, 2, \dots, 63$ .

This may be used to good effect in graphing by using

```
PRINT CHR$(192 + X); "+"
```

as in the following program to graph a portion of  $X = .062Y^2$  which lies in the first quadrant.

```
104 PRINT "GRAPH OF X = .062*Y*Y"
110 Y = 30
120 X = INT(.062*Y*Y)
130 PRINT X; CHR$(192 + X); "+"
150 Y = Y -2
160 IF X >2 THEN 120 ELSE 160
```

Note that it will be necessary to depress BREAK to get the computer's attention after the graph is plotted.(WHY)?

If we had not created the "tight loop" by using ELSE 160, then when X got to be  $\leq 2$  the display screen would have inserted

```
READY
```

```
>---
```

onto the screen and scrolled the top off the screen. If you are not sure of this,

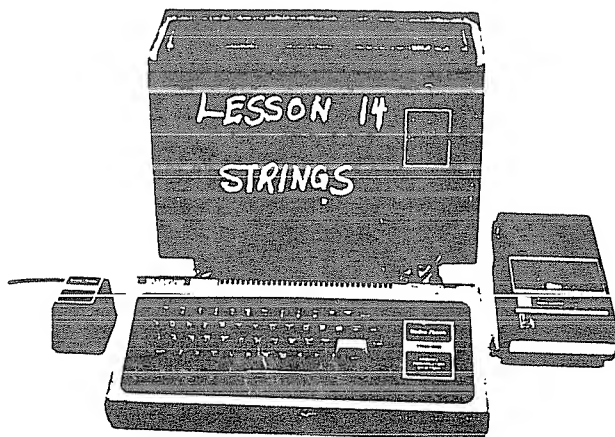
change 160 to 160 IF X >2 THEN 120

and reRUN the revised program.

The second edition of the LEVEL II BASIC Reference Manual (appendix C/6) shows a map of the graphic pixel blocks and their corresponding numbers. Both the character spacing (16 lines of 64 characters) numbered from 0 to 1023 and the pixel locations for use with SET (X,Y) (48 lines with  $0 \leq Y \leq 47$  of 128 pixels with  $0 \leq X \leq 127$ ) are shown mapped onto the grid. In each case the numbering starts in the upper left corner with 0 or (0,0).

If you remember the diagram on the right and use 128 + (pixels to be lighted), you will not need the reference.

1	2
4	8
16	32



### STRING FUNCTIONS

Let's re-examine the program of Lesson 4.

```

10  CLS
100 INPUT "PLEASE TYPE YOUR NAME"; N$
110 PRINT:PRINT:PRINT
120 PRINT "HELLO ";N$;". ":PRINT"      I HOPE YOU ARE ENJOYING THIS
                                         COURSE."

125 FOR Q = 1 to 500:NEXT Q
127 PRINT CHR$(23)
130 GOTO 110

```

Line 10 clears the screen.

Line 100 prints what is in quotes and then awaits your input to string variable N\$.

Line 110 prints 3 blank lines (i.e., moves the next output line down 3 rows).

Line 120 prints HELLO, followed by whatever you typed into N\$, and then on the next line prints

I HOPE YOU ARE ENJOYING THIS COURSE. with a nine-space indentation, if you allowed nine spaces.

Line 125 is a "timewaster". It forces the computer to count from 1 to 500 by ones before continuing.

Line 127 changes all output to double-size until the next CLS instruction is executed. This eliminates half of the material currently on the screen, but new material will be printed in double-wide characters.

Line 130 sends the program back to line 110 to print three more blank lines followed by HELLO your name (double-size this time) etc., etc.,..... until you depress the BREAK key to stop it.

That wasn't too hard was it?

Play around a bit with the messages in line 120 and change them to suit your whim--be sure to include N\$ as part of your output--note that N\$ is a variable and must be outside of the quotes, separated from the quotes by semicolons.

String variables use up quite a lot of memory space--not as much as on many computers, since the TRS-80 LEVEL II string variable uses only as much space as it needs--i.e. short strings use less memory than long strings. On many computers, string variables require the same (maximum) space even if not all is needed. If you have a 4-K Level II, you may run out of memory if you use many variable strings. The following program is designed to create very simplistic sentences and runs on 4-K. Note that string variables may be dimensioned like floating-point variables.

```

2  CLEAR 1000
5  CLS
10  DIM N$(7), V$(2)
100 PRINT "PLEASE TYPE 8 NOUNS."
110 FOR K = 0 TO 7
120   PRINT K + 1, :INPUT N$(K)
130 NEXT K
200 PRINT "TYPE 3 VERBS. PLEASE USE DIFFERENT TENSES."
210 INPUT "1 ";V$(0)
220 INPUT "2 ";V$(1)
230 INPUT "3 ";V$(2)
500 PRINT "THAT IS FINE. NOW I'LL CREATE SOME SIMPLE SENTENCES."
600 PRINT N$(RND(8)-1);" " ; V$(RND(3)-1);" ";N$(RND(8)-1);"."
620 GOTO 600

```

Put it on your computer and RUN it. If you should run out of memory (indicated by OM ERROR or OS ERROR) see if you can figure out what to do about it (see Lesson 11).

It is also feasible to store the words in DATA statements. In this case they are read into the program using a READ statement as the following program, designed with tongue-in-cheek to generate impressive phrases for use in reports and grant requests.

READ W\$(I) is like INPUT W\$(I), except that it obtains the string value for W\$(I) from a DATA statement instead of from the keyboard.

The individual string entries in the data statement are each enclosed in quotes and entries are separated by commas. The computer will go to the next DATA statement if the DATA statement on which it is working is exhausted. DATA may also read from tape—but not here.



DATA statements may also be used to store numerical data. In this case numerical variables (no \$) are used in the READ statement and quotes are not used in the DATA statement.

```

10 DIM W$(35)
20 FOR I = 0 TO 35
30   READ W$(I)
40 NEXT I
50 DATA "FUNDAMENTAL", "BASIC", "INTUITIVE", "STUDENT-CENTERED",
  "HOMOGENEOUS", "MODULAR"
51 DATA "PARALLEL", "CENTRAL", "EQUAL", "SUPERIOR", "COLLECTIVE",
  "JUDGMENTAL"
52 DATA "ACCOUNTABLE", "UNGRADED", "NON-SEXIST", "BEHAVIORIAL",
  "SCIENTIFIC", "HUMANITARIAN"
53 DATA "INTRINSIC", "FOUNDATION", "SUPPORTIVE", "LIMITED", "UNIFORM",
  "DISADVANTAGED"
54 DATA "STRUCTURE", "PERFORMANCE", "REINFORCEMENT", "GROUPING",
  "CURRICULUM", "ENVIRONMENT"
55 DATA "OBJECTIVE", "BOARD", "EXPERIMENT", "POLICY", "TEACHING",
  "EXAGGERATION"
100 PRINT W$(RND(12)-1);" ";
110 PRINT W$(RND(12)+11);" ";
120 PRINT W$(RND(12)+23)
130 PRINT
140 FOR Q = 1 TO 600 : NEXT Q
150 GOTO 100

```

It is easy to substitute words of the reader's choice by changing the DATA statements (50,51,52,53,54,55). The program chooses its first word from the first 12 words of the data list, the second word from words 13-24 and the third word from words 25-36 in the DATA list. You may be interested in seeing what happens if you replace one of the words between quotes with a blank space between quotes.

A SECOND SENTENCE GENERATOR

Let's write a program to generate more sentences. Consider the sentence:

HELEN QUICKLY SAW THE RED FOX.

The parts of speech are

NOUN ADVERB VERB ARTICLE ADJECTIVE NOUN

Another sentence having the same pattern is:

JOHN SADLY SWEEP THE DIRTY FLOOR.

Let's write a program that will accept 10 nouns, 5 adverbs, 6 verbs and 7 adjectives. We shall start by setting aside the needed space by using CLEAR and DIMension statements.

```

2 CLEAR 1000
5 CLS
10 DIM NS(9), AS(4), VS(5), JS(6)
20 SS = " "
```

Next, we shall write a portion of the program to permit us to type in the words of our choice.

```

100 PRINT "TYPE TEN NOUNS. DEPRESS (ENTER) AFTER EACH WORD."
110 FOR K = 0 TO 9
120   PRINT K+1;
130   INPUT NS(K)
140 NEXT K
150 PRINT
```

That wasn't so hard was it? The computer numbers the stored words 0 to 9, but humans seem to prefer 1 to 10, so that is what we used for each. Let's continue.

```

200 PRINT "PLEASE TYPE 5 ADVERBS. DEPRESS (ENTER) AFTER EACH WORD."
205 PRINT "ADVERBS USUALLY END IN -LY."
210 FOR K = 0 TO 4
220   PRINT K + 1;
230   INPUT AS(K)
240 NEXT K
250 PRINT
300 PRINT "PLEASE TYPE 6 VERBS (PAST TENSE). DEPRESS (ENTER) AFTER EACH WORD."
310 FOR K = 0 TO 5
320   PRINT K + 1;
330   INPUT VS(K)
340 NEXT K
350 PRINT
```

*continued on next page*

```

400 PRINT "PLEASE ALSO TYPE 7 ADJECTIVES.  DEPRESS (ENTER) AFTER
      EACH WORD."
410 FOR K = 0 TO 6
420   PRINT K+1
430   INPUT J$(K)
440 NEXT K
450 PRINT
500 PRINT "THAT IS FINE .  NOW, I'LL CREATE SOME SENTENCES FOR YOU."
600 FOR Q = 1 TO 1000 : NEXT Q
610 PRINT
620 PRINT  N$(RND(10)-1); S$; A$(RND(5)-1); S$; V$(RND(6)-1); S$; "THE"
      ; S$; J$(RND(7)-1); S$; N$(RND(10)-1); " . "
630 GOTO 600

```

RUN ENTER

A list of the major **STRING** instructions follows:

Function	Operation	Examples
ASC( <i>string</i> )	Returns ASCII code of first character in string argument.	ASC(B\$) ASC("H")
CHR\$( <i>code exp</i> )	Returns a one-character string defined by <i>code</i> . If <i>code</i> specifies a control function, that function is activated. (See Lesson 4 for equivalent list.)	CHR\$(34) CHR\$(K)
FRE( <i>string</i> )	Returns amount of memory available for string storage. Argument is a <i>dummy</i> variable.	FRE(A\$)
INKEY\$	Strobes keyboard and returns a one-character string corresponding to key pressed during strobe (null string if no key is pressed). Usually used as	INKEY\$
300 A\$ = INKEY\$ : IF A\$="" THEN 300 ELSE PRINT A\$;		
<p>NOTE: There is <u>no</u> space between the quote marks. Thus IF A\$ is the null string, the program loops back to strobe the keyboard again. ELSE it prints the value typed and continues with the next instruction.</p>		
LEN( <i>string</i> )	Returns length of <i>string</i> (zero for null string).	LEN(A\$+B\$) LEN("HOURS")

Function	Operation	Examples
LEFT\$( <i>string</i> , <i>n</i> )	Returns first <i>n</i> characters of <i>string</i> .	LEFT\$(A\$,1) LEFT\$(L1\$+C\$,8) LEFT\$(A\$,M+L)
MID\$( <i>string</i> , <i>p</i> , <i>n</i> )	Returns substring of <i>string</i> starting at position <i>p</i> in <i>string</i> and containing the next <i>n</i> characters.	MID\$(M\$,5,2) MID\$(M\$+B\$,P,L-1)
RIGHT\$( <i>string</i> , <i>n</i> )	Returns last <i>n</i> characters of <i>string</i> .	RIGHT\$(NA\$,7) RIGHT\$(AB\$,M2)
STR\$( <i>numeric exp</i> )	Returns a string representation of the evaluated argument. This converts numeric variables to string variables.	STR\$(1.2345) STR\$(A+B*2)
STRING\$( <i>n</i> ," <i>char</i> ")	Returns a sequence of <i>n</i> " <i>char</i> " symbols using first character of <i>char</i> . Useful in creating borders or divisions of the output.	STRING\$(30,".") STRING\$(25,"A") STRING\$(5,C\$)
VAL( <i>string</i> )	Returns a numeric value corresponding to a numeric-valued string. This is the inverse of STR\$( ).	VAL("1"+A\$+"."+C\$) VAL(A\$+B\$) VAL(G1\$)

(*string* may be a string variable, expression, or constant.)

To concatenate strings (put them together), use the + operator as in

"2" + "YOU" + "76" = 2YOU76

Note that it is important to include desired leading and trailing spaces inside the quotes if you wish them to appear both in concatenation and in PRINT "YOUR NAME IS ";N\$; " ."

A CIPHER PROGRAM

The following program creates a simple, but reasonably secure cipher by first changing each character into its numerical ASCII equivalent, using the ASC( ) instruction, adding a computer-generated random number to the ASCII code, moving the result to the desired range  $48 \leq N \leq 90$ , and then changing the modified ASCII code into an appropriate symbol using CHR\$( ). Decipherment is done using the reverse process. You may change the basic key by changing instruction 135 to  $R = Q * R$  for some other appropriate Q-value. But don't make the change unless you understand random-number generators. The choice of Q is critical and depends upon the number system used.

```

10 DIM C(500)
40 L = 0
50 INPUT "PLEASE TYPE YOUR KEY--POSITIVE TO ENCIPHER, NEGATIVE TO DE-
    CIPHER.";K
60 R = ABS(K) : R1 = R
70 PRINT "PLEASE TYPE COMPLETE MESSAGE FOLLOWED BY THE SYMBOL +."
100 P$ = INKEY$ : IF P$ = "" THEN 100 ELSE PRINT P$;
105 IF P$ = "/" THEN 100
112 IF P$ = "+" THEN 200
115 L = L+1
117 IF K < 0 THEN R1 = -1*R1
120 C(L) = ASC(P$)+R1
125 IF C(L) > 90 THEN C(L) = C(L) - 43
130 IF C(L) < 48 THEN C(L) = C(L) + 43
135 R = 197*R
137 R = R-100000*INT(R/100000)
139 R1 = INT(.004*R)
150 GOTO 100
200 PRINT
205 IF K < 0 THEN 300
210 FOR M = 0 TO L-1 STEP 5
215   FOR N = 1 TO 5
220     PRINT CHR$(C(M+N));
230   NEXT N
240   PRINT " ";
250 NEXT M
260 END
300 FOR M = 1 TO L
310   PRINT CHR$(C(M));
320 NEXT M
400 END

```

Note, that instruction 100 uses the INKEY instruction which strobes the keyboard once. If a key is depressed during the strobe, that value is stored in variable P\$, otherwise P\$ contains a null (not a blank, just nothing at all). The instruction

IF P\$ = "" THEN 100 ELSE PRINT P\$

sends the computer back to 100 P\$ = INKEY\$ if P\$ is blank, otherwise it prints the value of P\$ (the key just depressed) on the screen and continues to the next instruction

```
105 IF P$ <="/" THEN 100
```

which essentially refuses any symbols before / (= 47 in ASCII--see Lesson 4).

```
112 IF P$ = "+" THEN 200
```

sends the computer to 200 when the message is complete, as indicated by typing

Lines 115 to 150 encipher or decipher the message depending upon whether the key K is positive (encipher) or negative(decipher). Each user will presumably have a different numerical key.

```
205 IF K < 0 THEN 300
```

sends the computer to 300 if it is deciphering the message, and prints the entire message as one string (blanks between words were not enciphered--they could have been, but weren't).

IF K >= 0 then the computer continues with instruction 210 to 260 which displays the enciphered message in blocks of five symbols (a standard cipher practice).

### ROBOT COUNSELOR

The following program, which runs on any 4K TRS-80, gives a brief insight into what programs that have more memory available can be expected to do.

```
1 REM      ROBOT COUNSELOR
2 RANDOM
5 CLS
7 CLEAR 100
10 DIM G$(9)
11 G$(0) = "HMM...MMM...,VERY INTERESTING"
12 G$(1) = "STRANGE. NOT ABNORMAL, YOU UNDERSTAND, JUST STRANGE"
13 G$(2) = "FASCINATING"
14 G$(3) = "UNUSUAL, BUT PERFECTLY REASONABLE UNDER THE CIRCUMSTANCES"
15 G$(4) = "THERE MAY EASILY BE MORE TO IT THAN THAT"
16 G$(5) = "THAT MAY BE AN EXAGGERATION"
17 G$(6) = "VERY COMMON IN TODAY'S WORLD"
18 G$(7) = "BEWILDERING. I DO NOT UNDERSTAND IT"
19 G$(8) = "I HOPE TOMORROW WILL BE BETTER"
20 G$(9) = "SURPRISING IN THE LIGHT OF YOUR BACKGROUND"
50 PRINT "I'D LIKE TO HELP YOU. MAYBE TOGETHER WE CAN WORK OUT YOUR
   PROBLEMS. PLEASE ANSWER ME WHEN I ASK YOU QUESTIONS."
```

*continued...*

```

52 FOR Q = 1 TO 400 : NEXT Q
53 PRINT
55 PRINT "JUST TYPE YOUR ANSWERS, THEN DEPRESS THE WHITE (ENTER) KEY."
57 FOR Q = 1 TO 300 : NEXT Q
59 PRINT
60 INPUT "PLEASE TYPE YOUR FIRST NAME AND DEPRESS (ENTER) KEY. "; N$
61 PRINT : PRINT : PRINT "HI "; N$
70 PRINT : PRINT
80 FOR Q = 1 TO 900 : NEXT Q
100 PRINT : PRINT "HOW DO YOU FEEL, "; N$? PLEASE ANSWER IN ONE OR TWO
WORDS."
110 INPUT R$
115 FOR Q = 1 TO 500
120 PRINT:PRINT:PRINT "OH, I SEE. THAT IS "; G$(RND(10)-1);"."
130 FOR Q = 1 TO 800 + RND(1000) : NEXT Q
139 PRINT
140 PRINT "WHO DO YOU THINK MAKES YOU FEEL "; R$;"?"
150 INPUT R2$
155 IF R2$ = "ME" THEN R2$ = "YOU ALONE"
156 IF R2$ = "MYSELF" THEN R2$ = "YOU YOURSELF"
157 IF R2$ = "YOU" THEN R2$ = "THE TRS-80"
160 PRINT:PRINT
163 FOR Q = 1 TO 100 + RND(500):NEXT Q
165 PRINT "WELL NOW, "; N$; ", THAT IS CERTAINLY "; G$(RND(10)-1);"."
170 FOR Q = 1 TO 500 + RND(600):NEXT Q
180 PRINT:PRINT "DO YOU REALLY BELIEVE "; R2$;"MAKES YOU FEEL "; R$;"?"
190 INPUT A$
200 IF A$ = "NO" THEN PRINT "WELL "; N$; ", PLEASE BE FRANK WITH ME. IT
SEEMS "; G$(RND(10)-1);"." GOTO 140
210 PRINT:PRINT:
220 FOR Q = 1 TO 10 + RND(1000):NEXT Q
225 PRINT "THAT IS "; G$(RND(10)-1);"." IT IS "; G$(RND(10)-1);"."
228 PRINT "THINK ABOUT IT A BIT."
229 PRINT:PRINT "IT IS INDEED "; G$(RND(10)-1);"." :PRINT:PRINT
230 PRINT "REALLY NOW, TELL THE TRUTH. PLEASE BE FRANK WITH ME."
235 PRINT "IT SEEMS "; G$(RND(10)-1);"."
240 GOTO 100

```

After you have run the program a few times, change some of the phrases stored in G\$( ) in lines 11 to 20. See if you can produce a more interesting ROBOT COUNSELOR program.

LOGICAL OPERATORS

Have you wondered how the computer handles IF instructions?

The instruction

```
IF X >13 THEN 420 ELSE 30
```

examines the number stored in location X and if X is greater than 13, sends the computer to statement 420 for its next instruction but if X is less than or equal to 13 sends the computer to statement 30 for its next instruction.

The instruction

```
IF X >125 THEN X = 125
```

examines the variable X and if X is greater than 125, sets X = 125.

Actually the expression  $X > 125$  is evaluated as a logical expression.

If the current value of X makes  $X > 125$  a TRUE statement, then the value of the logical expression

$(X > 125)$  is -1.

If the current value of X makes  $X > 125$  a FALSE statement, then the value of the logical expression  $(X > 125)$  is 0.

You can examine this by running the program

```
100 Y = 25
110 FOR K = 1 TO 8
120 X = K * Y
130 PRINT K,X,(X > 125)
140 NEXT K
```

which produces the output

1	25	0
2	50	0
3	75	0
4	100	0
5	125	0
6	150	-1
7	175	-1
8	200	-1

The actual values (TRUE = -1, FALSE = 0) used are unimportant at this stage, and may vary on different computers. (LEVEL I TRS-80's use TRUE = 1, FALSE = 0 .)



If the value produced represents TRUE (i.e., -1) then the remaining statements on the IF line are carried out (possibly several statements separated by colons).

If the value produced represents FALSE (i.e., 0) then the computer skips the rest of the instructions in that statement line (possibly several instructions) and goes to the next line or the ELSE statement for its next instruction.

It is possible to combine statements using logical operators:

AND      If both expressions are true    TRUE (= -1) otherwise FALSE (= 0)

OR        If either expression is true    TRUE (= -1) otherwise FALSE (= 0)

NOT       Interchanges TRUE (-1) and FALSE (0).

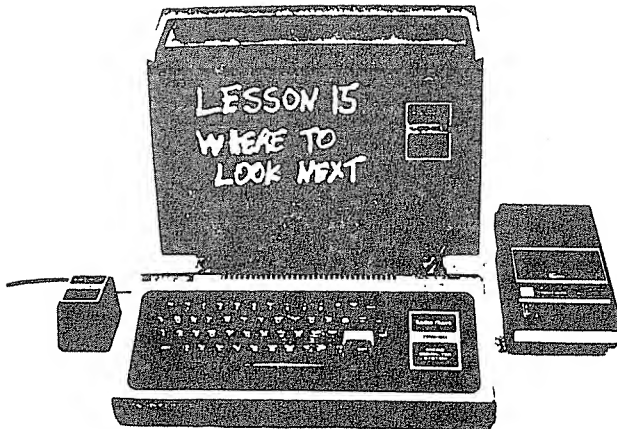
EXAMPLE:

```

100 INPUT A,B
110 PRINT "A=";A, "B=";B
120 IF A > B THEN PRINT "A IS GREATER THAN B"
130 IF (A > 0) AND (A < 10) THEN PRINT " 0 < A < 10." ELSE PRINT "A <= 0 or
    A = 10."
140 IF (B > 20) OR (B < -20) THEN PRINT "ABS (B) > 20."
150 IF ((A > 0) AND (B > 0)) OR ((A < 0) AND (B < 0)) THEN PRINT "A*B > 0."
    ELSE PRINT "A*B <= 0."
160 GOTO 100

```

Actually the AND, OR, NOT logical operations also may be used to test individual bits in the binary storage of TRS-80 words since the operations perform Boolean operations on the bits of numbers as well as on logical statements. Readers interested in such masking operations should first consult chapter 8 of the TRS-80 Level II BASIC REFERENCE MANUAL and then a text on Boolean (logical) algebra.



#### WHERE TO LOOK FOR ADDITIONAL INFORMATION

Your first source of additional information is the LEVEL I or LEVEL II BASIC REFERENCE MANUAL that came with your TRS-80. You should also be receiving (free) the Radio Shack Micro Computer Newsletter if you filled out the card requesting it that came with your TRS-80.

Fellow computer buffs are another excellent source of information. If you are in real difficulty, don't hesitate to telephone or write to The Radio Shack Computer Services Center in Ft. Worth. They maintain a staff of knowledgeable people who seem both willing and able to help TRS-80 owners. The address is

Computer Services  
900 Two Tandy Center  
Ft. Worth, TX 76102

Telephone:  
1-(800)-433-1679

They do not assist in debugging or writing programs, but will gladly assist you if you have questions about what a given BASIC or Z-80 instruction does or about the TRS-80 hardware.

Most computer programmers, either hobbyists or professionals, learn most about computing by burning midnight oil at the computer. One learns to compute by computing, and then by analyzing the results.

There seem to be clubs, conventions and magazines devoted to almost any hobby that interests you, and computers are no exception. There are even special clubs, conventions and magazines devoted exclusively to microcomputers. Some are so highly specialized that they concentrate on the TRS-80 microcomputer. Ask around your own area and see what micro-

computer clubs and/or conventions are available. If no microcomputer club exists in your area, organize one yourself.

No list of microcomputer magazines (journals) can ever be complete—new ones seem to start up almost monthly; others drop into oblivion. Some are excellent; others are a waste of funds. Most will send you a sample copy if you ask for one, then you can judge how well it fits your particular interests.

Here are some that we take, with our favorites indicated by a star ★. Prices were current at time of printing.

#### Micro Computer Conference Proceedings

The Best of the West Coast Computer Fair, 333 Swett Road, Woodside, CA, 94062. Set of 4 \$53; Vols I (1976) & III (1978) \$13.72 ea.; Vols II (1977) & IV (1979) \$14.78 ea.

- ★ Show & Tell microCOMPUTER Conference Proceedings, Department of Mathematics, University of Oklahoma, Norman, OK, 73019. (1978) \$7.50; (1979) \$10; (1980) \$10; (1981) \$10. Conference held in May each year.

#### Micro Computer Periodicals

- ★ Byte, Byte Subscriptions, P.O. Box 590, Martinsville, NJ, 08836, monthly, \$15 per year. Both hardware and programs. Some technical material.

Computronics, TRS-80 Monthly News Magazine, P.O. Box 149, New York City, NY, 10956, monthly, \$24 per year. Not worth the cost in my judgment—request an examination copy and judge for yourself. Magazines change.

The Computing Teacher, Computing Center, Eastern Oregon State College, La Grande, OR, 97850, bi-monthly, \$8 per year.

- ★★ Creative Computing, P.O. Box 789-M, Morristown, NJ, 07960, monthly, \$15 per year. Probably the best buy available. Something for everyone. Well written and authoritative articles.

Data General News, Data General Corporation, Southboro, MA, 01772 (free—devoted to their computers).

Dr. Dobbs Journal of Computer Calisthenics & Orthodontia, The Peoples Computer Co., Box E, Menlo Park, CA, 94025, monthly, \$12 per year. Advanced and excellent.

- ★ Games, Games Productions, Inc., 515 Madison Ave., New York City, NY, 10022, bi-monthly, \$5.97 per year. Not computer oriented, but lots of good ideas you can use.

Games & Puzzles, Edu-Games (U.K.) Ltd., P.O. Box 4, London, England, N64DF

- ★ Journal of Recreational Mathematics, Baywood Publishing, 120 Marine, Farmingdale, NY, 11735. Interesting problems, some computer related. Quarterly, \$25. (\$10 to individuals, home address, personal check.)
- Personal Computing, Circulation, Personal Computing, 1050 Commonwealth Ave., Boston, MA, 02215, monthly, \$14 per year.
- ★ Computer Music Journal, ISSN 0148-9276 MIT Press Journals, 28 Carleton St., Cambridge, MA 02142. Price \$4 per issue, \$20 yearly.
- ★ PROG 80, Softside Publications, 17 Briar Cliff Dr., Milford, NH, 02055, monthly, \$15 per year. Lots of goodies for the serious TRS-80 programmer.
- Radio Shack Microcomputer Newsletter, Radio Shack, One Tandy Center, Fort Worth, TX, 76102, TRS-80 owners receive free for 6 months, then \$12/year.
- Recreational Computing, People's Computer Company, 1263 El Camino Real, Box E, Menlo Park, CA, 04025, 6 issues per year, \$10.00.
- The Recreational Programmer, Computer Software, P.O. Box 2571, Kalamazoo, MI, 49003, bi-monthly, \$12 per year.
- Softside, Softside Publications, 17 Briar Cliff Dr., Milford, NH, 02055, monthly, \$15 per year. Lots of BASIC games in this one.
- T-PAL, The Mail Mart, Box 11102, San Francisco, CA, 94101, monthly, \$24 per year. Seems expensive for what you get. Ask for a sample copy, it should have improved.
- ★ TRS-80 Computing, Computer Information Exchange, P.O. Box 158, San Luis Rey, CA, 92068, monthly, \$15 per year. Lots of goodies here.
- ★ 80-US, 80-NW Publishing, 3220 N. 32nd St., Tacoma, WA, 98407, sample copy, \$3, \$16 per year. Six issues annually.
- PAGE, Bulletin of the Computer Arts Society, John Lansdown, 50151 Russell Square, London, WC1B4JX, England, quarterly, \$51 per year (students, half-price). Or write Kurt Lauckner, Math. Dept., Eastern Michigan University, Ypsilanti, MI, 48917. Devoted to computer assisted art.
- TRS-80 Users Group, 7554 Southgate Rd., Fayetteville, NC, 28304, monthly, \$12 per year. An informal publication for amateurs that is well worth the price.
- ★ 80 Microcomputing, Started in January 1980 and so far looks to be well worth the \$12/year. This monthly slick magazine devoted to TRS-80 does not pull punches and keeps you informed on the latest updates and changes and evaluations of hardware and software related to the TRS-80.

### Where to Buy Programs

We don't purchase many programs; we write our own because it is more fun. Most of the microcomputer magazines and conference proceedings carry programs. Some, you can obtain on tape at a modest cost. Most carry ads for programs that will run on your TRS-80. Some sell programs that are definite "rip-offs". Others are bargains. Few have adequate documentation with them. Still, you will almost certainly be tempted to buy some, just to try them out. Read the ads. Go to your local Radio Shack (or Radio Shack Computer Center, if you have one...many middle-to-large cities do). Go to any other computer stores in your area and ask to see the programs in action. (Don't let your local computer store disparage the TRS-80. Some will because they do not sell them.) Attend a microcomputer club in your area. Ask about available software.

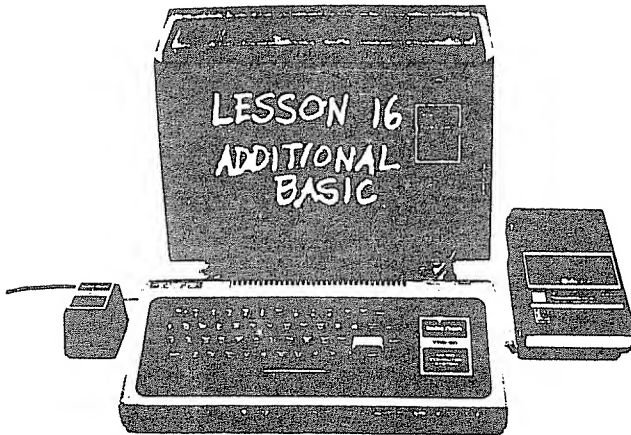
If you haven't seen Leo Christopherson's Android Nim (with sound), you should. Nim isn't much of a game, but the animation is superb (1980 price, \$14.95, runs on 16K Level II TRS-80). If you wish a TRS-80 chess or checker playing program, look over the results from the most recent national microcomputer chess or checkers tournament. There are a dozen programs available for under \$20 each, and the quality varies considerably. Read the comments and the ads in whatever microcomputer periodicals you and/or your library take.

You should probably write to:

People's Software  
Computer Information Exchange  
Box 158  
San Luis Rey, CA 92066

They put a couple of dozen programs on a tape and sell it for \$8.00 (see Lesson 7 ). They also sell more expensive programs, like People's Pascal ( \$ 23.50). New programs are added frequently. Drop them a card asking for a current price list.

HAVE FUN! GROW WITH YOUR NEW HOBBY. READ, EXPERIMENT, DISCUSS YOUR EXPERIENCES. HAVE FUN!



Your TRS-80 Level II BASIC Reference Manual contains details of the available Level II BASIC instructions. If you have an expansion interface, line printer, or disk you will find additional instructions defined in the accompanying reference manuals. Here are several additional Level II BASIC instructions you may find convenient.

#### Additional Keyboard Commands

#### Examples

AUTO *mm*,*nn*

Turns on automatic line numbering beginning with *mm* and in steps of *nn*. If *nn* is omitted, the default step size is 10.

AUTO 100

AUTO 100,5

AUTO 300, 20

DELETE *mm-nn*

Deletes program steps from line *mm* to line *nn* inclusive. Both *mm* and *nn* should be actual statement numbers in the program.

DELETE 410

DELETE 80-230

LIST *mm-nn*

Lists program from line *mm* to line *nn*.

LIST 270-400

LIST 140-

LIST -270

LIST

RUN *mm*

Executes program beginning at line *mm*.

RUN

RUN 450

SYSTEM

Enter monitor mode for loading Z-80 language program from cassette tape. Frequently used with advanced commercial programming.

SYSTEM

Additional Program InstructionsTRON

Turns on TRACE program in which variable values are shown on screen to help you debug program. TRACE program is always in the TRS-80, as part of your Level II BASIC ROM.

TROFF

Turns TRACE program off. The TRACE program is still in the computer.

USR(*n*)

Branches to a Z-80 machine language subroutine already entered by the user. You'll need to use POKE first.

RESTORE

Resets the DATA pointer to the first DATA statement to permit program to reuse the same data.

GOSUB *nn*

Branch to the subroutine beginning at line number *nn*.

RETURN

Each subroutine must contain at least one RETURN statement. This sends the program back to the line following the GOSUB that branched to the subroutine.

ON  $\alpha$  GOTO *mm*, *nn*, *pp*, *qq*, '''

If the value of INT( $\alpha$ ), where  $\alpha$  is a variable or expression, is one of the numbers 1,2,3,4, ..., *k* then go to the statement number in that position on the list of statement numbers *mm*, *nn*, *pp*, *qq*, '''.  
 -

ON  $\alpha$  GOSUB *mm*, *nn*, *pp*, *qq*, '''

Same as ON  $\alpha$  GOTO except the RETURN pointer is set so program will return to statement following ON  $\alpha$  GOSUB.... when RETURN is encountered.

ExamplesTRONTROFFUSR(0)RESTOREGOSUB 460GOSUB 3700RETURNON Z GOTO 200, 300, 400ON K+2 GOTO 700,700,400,300,500ON K GOSUB 3300, 2000, 400

## Arithmetic Functions

Function	Operation(unless noted otherwise, <u><math>-1.7E+38 \leq \text{exp} \leq 1.7E+38</math></u> )	Examples
ABS( <i>exp</i> )	Returns absolute value.	ABS(L*.7) ABS(SIN(X))
ATN( <i>exp</i> )	Returns arctangent in radians.	ATN(2.7) ATN(A*3)
CDBL( <i>exp</i> )	Returns double-precision representative of <i>exp</i> .	CDBL(A) CDBL(A+1/3#)
CINT( <i>exp</i> )	Returns largest integer not greater than <i>exp</i> . Limits: $-32768 \leq \text{exp} < +32768$ .	CINT(A#+B)
COS( <i>exp</i> )	Returns the cosine of <i>exp</i> ; assumes <i>exp</i> is in radians.	COS(2*A) COS(A/57.29578)
CSNG( <i>exp</i> )	Returns single-precision representation, with 5/4 rounding in least significant decimal when <i>exp</i> is double-precision.	CSNG(A#) CSNG(.33*B#)
EXP( <i>exp</i> )	Returns the natural exponential, $e^{\text{exp}} = \text{EXP}(\text{exp})$ .	EXP(34.5) EXP(A*B*C-1)
FIX( <i>exp</i> )	Returns the integer equivalent to truncated <i>exp</i> (fractional part of <i>exp</i> is chopped off).	FIX(A-B)
INT( <i>exp</i> )	Returns largest integer not greater than <i>exp</i> .	INT(A+B*C)
LOG( <i>exp</i> )	Returns natural logarithm (base e) of <i>exp</i> . Limits: <i>exp</i> must be positive.	LOG(12.33) LOG(A+B*B)
SGN( <i>exp</i> )	Returns $\pm 1$ for negative <i>exp</i> ; 0 for zero <i>exp</i> ; $+1$ for positive <i>exp</i> .	SGN(A*B+3) SGN(COS(X))
SIN( <i>exp</i> )	Returns the sine of <i>exp</i> ; assumes <i>exp</i> is in radians.	SIN(A/B) SIN(90/57.29578)
SQR( <i>exp</i> )	Returns square root of <i>exp</i> . Limits: <i>exp</i> must be non-negative.	SQR(A*A - B*B)
TAN( <i>exp</i> )	Returns the tangent of <i>exp</i> ; assumes <i>exp</i> is in radians.	TAN(X) TAN(X*.01745329)

*exp* may be any numeric variable, constant or computed expression.



There are a number of other Level II BASIC instructions available for special use on your TRS-80. If your programming is advanced enough to need these extra instructions see the TRS-80 Level II BASIC Reference Manual and the reference manuals accompanying any additional equipment you may have (Disk, Lineprinter) etc.

Three examples are:

#### POKE *location, value*

Loads *value* into *memory location*. The arguments must be decimal numeric variables, constants or functions with  $0 \leq \text{value} \leq 255$ . Used in advanced programming to change memory content or display screen and with `USR( )` function. See Chapter 8 of Level II BASIC Reference Manual.

POKE 16396,23

will disable the `BREAK` key. This is sometimes used to prevent unsophisticated users from `BREAK`-ing into a program.

#### PEEK (*address*)

Returns the (decimal) value stored in the memory *address* specified. Also, makes values of Z-80 machine language variables available to BASIC programs. See Chapter 8 of Level II BASIC Reference Manual.

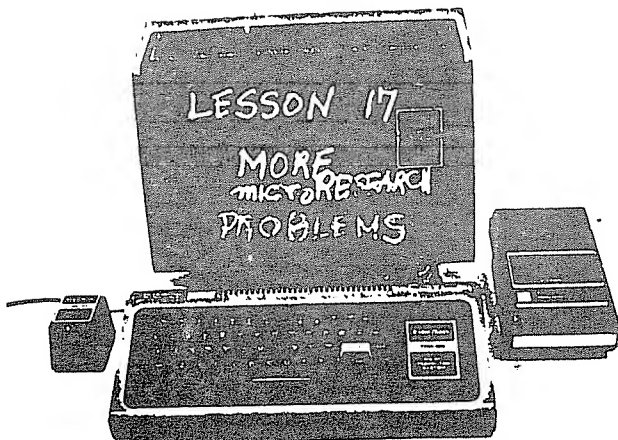
```
10 FOR K = 0 TO 12288
20 PRINT K;PEEK(K),
30 NEXT K
```

will display the location and contents of your Level II BASIC ROM. It won't mean much to you, but it will do it.

#### RANDOM

is frequently executed once near the beginning of a program to reseed the `RND( )` instruction. Do not use `RANDOM` inside of a loop or where it will be used repeatedly in your program.

```
1 REM      program name
2 RANDOM
```

EXAMPLE 17-1

Let us examine a program designed to determine the factors of a given integer. Forms of this program have appeared in several magazines, books and even in tape form. The given program works. It produces the desired factors for any positive integer  $Z$ . However, it is easy to improve upon the program, as we shall see.

First examine the original program to see why it works.

```

10  REM  FIRST WORKING ATTEMPT TO FACTOR A NUMBER
100  INPUT "TYPE NUMBER TO BE FACTORED"; Z
110  PRINT "THE PRIME FACTORS OF";Z;"ARE";
120  N = Z
130  F = 2
140  IF N/F < > INT (N/F) THEN 200
150  PRINT F;
160  N = N/F
170  IF N = 1 THEN STOP ELSE 140
200  F = F + 1
210  IF F <= Z THEN 140

```

The heart of the program is instruction 140

IF  $N/F < > \text{INT}(N/F)$  THEN 200

which uses the integer function  $\text{INT}()$  to determine whether or not  $N/F$  is an integer (i.e., whether or not  $F$  is a factor of  $N$ ). If  $F$  is not a factor of  $N$ , the program jumps to instruction 200 which increases  $F$  by 1. If new  $F \leq Z$  the program loops back to instruction 140 to determine whether or not the new  $F$  is a factor of  $N$ . If  $F$  is a factor of  $N$ , the value of  $F$  is printed,  $N$  is replaced by  $N/F$  and if the new  $N = N/F < > 1$ ,

the test is repeated. When either of the conditions  $N = 1$  or  $F > Z$  occurs, the original number  $Z$  is completely factored.

The program also uses an axiom many experienced programmers have found worthwhile

Don't alter the value of  
an INPUT variable inside your program

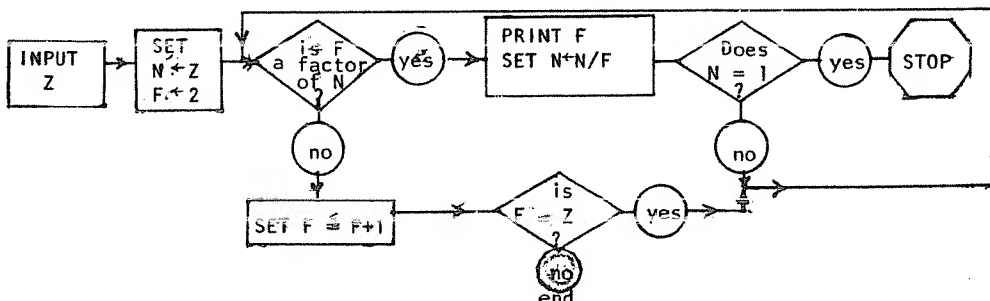
Note that the program inputs the value of  $Z$  but then sets  $N = Z$  and works with  $N$ . We could have ignored this motto here, but many experienced programmers will tell you they stick to it in their own work. It avoids trouble.

The program produces the prime factors of any positive integer of six or fewer digits you put in. Try it and see. Then examine the program to see exactly how it works.

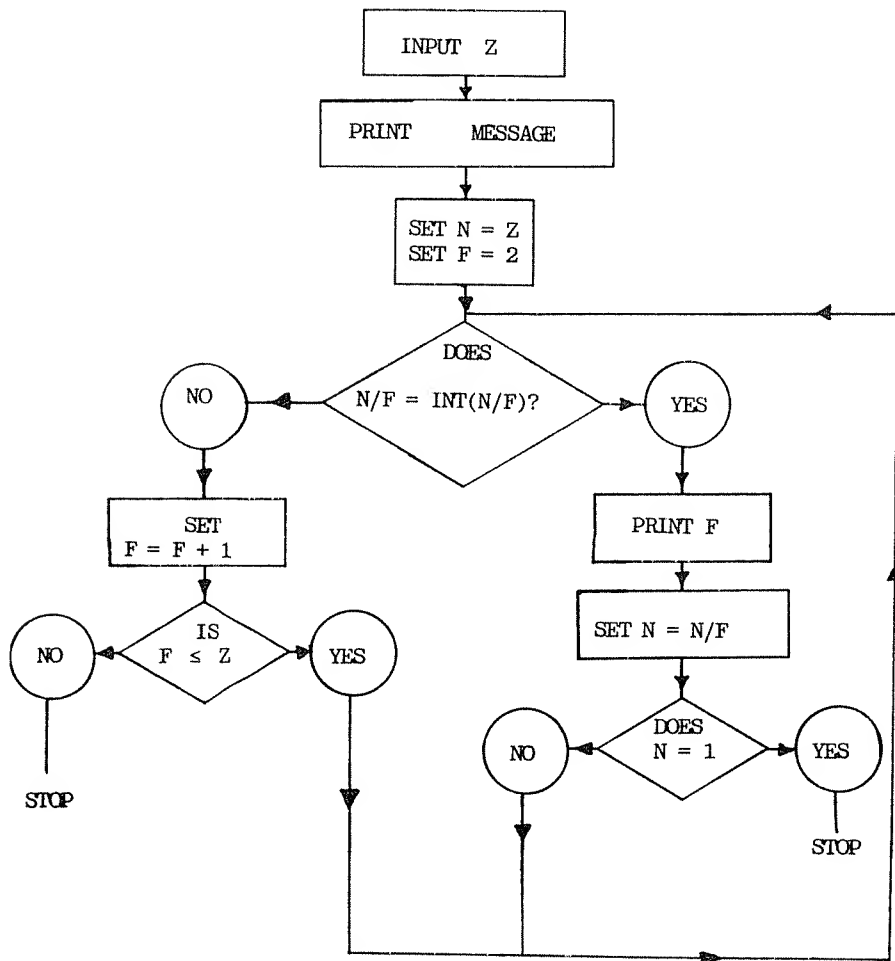
If you have any trouble following the program, make a chart of variables in the program and step the program through. If your INPUT value is  $Z = 12$ , your chart will contain

Z	N	F	PRINTED OUT
12			The prime factors of 12 are
	12	2	2
	6	2	2
	3	3	3
	1	STOP	

After you understand how the program works, you may wish to create a flowchart of the logic involved. Flowcharts are frequently used to express the logic involved in more complicated problems. This particular program is probably not involved enough to require a flowchart, but it is easy to "see" the flowchart of a simple problem.



Your flow chart should look something like this:



Note how easy it is to explain what the program is doing using a flow chart.

Actually, although the program in question has been published in several versions, it is an example of deplorably poor programming practice. It works fairly well on small values, but note the difference between  $Z = 1847$  and  $Z = 1848$  in the time required. Try it yourself before continuing.

Furthermore,  $Z = 1950$  takes :01 second to factor and  $Z = 1951$  takes 1:01 seconds to factor. This seems a long time for a powerful computer. Of course, it does do a lot of arithmetic in that time. In fact, that is just the trouble -- it does too much arithmetic. The program tests  $F = 2, 3, 4, 5, 6, 7, 8, 9, \dots, Z$  repeatedly. However, if we factor out all the 2's at the beginning, then there is no reason to also try 4, 6, 8, 10, etc. This should enable us to almost cut the time by half.

This can be done by making the following changes:

<u>OLD PROGRAM</u>	<u>FIRST REVISION</u>
10 REM FIRST WORKING ATTEMPT TO FACTOR A NUMBER	10 REM FIRST WORKING ATTEMPT TO FACTOR A NUMBER
100 INPUT "TYPE NUMBER TO BE FACTORED " ; Z	100 INPUT "TYPE NUMBER TO BE FACTORED " ; Z
110 PRINT "THE PRIME FACTORS OF " ; Z ; "ARE " ;	110 PRINT "THE PRIME FACTORS OF " ; Z ; "ARE " ;
120 N = Z	120 N = Z
130 F = 2	▶ 125 IF N/2 <> INT(N/2) THEN 135
	▶ 130 PRINT " 2";
	▶ 132 N = N/2
	▶ 134 GOTO 125
	▶ 135 F = 3
140 IF N/F <> INT(N/F) THEN 200	140 IF N/F <> INT(N/F) THEN 200
150 PRINT F;	150 PRINT F;
160 N = N/F	160 N = N/F
170 IF N = 1 THEN STOP ELSE 140	170 IF N = 1 THEN STOP ELSE 140
200 F = F + 1	▶ 200 F = F + 2
210 IF F <= Z THEN 140	▶ 210 IF F <= N THEN 140

In instruction

210 IF F <= Z THEN 140

there is really no reason to try F values that are larger than the current value of N (instead of the original Z), so we replace instruction 210 with

210 IF F <= N THEN 140

This should cut our time even further. Make the above changes in 125,

130, 132, 134, 135, 200, and 210. Then, try the program again.

N	Old Program Time	New Program Time
12	:01	:005
1847	:58	:30
1848	:01	:01
12345	:26	:14
-6	:01 <i>ignores sign</i>	:005
8.2	Runs through 210 and stops.	

Even the above program would not satisfy most professional programmers. The biggest complaints would probably be that the program malfunctions if negative values are used, and that the data was not tested to be sure the input value was an integer.

The problem of fractional values is readily overcome by inserting:

```
105 IF Z <> INT(Z) THEN PRINT "PLEASE USE INTEGER VALUES" : GOTO 100
```

Negative values could also be handled as improper input, but since they can be factored, we prefer another solution.

```
115 PRINT "(" ; SGN(Z) ; ")" ,
120 N = ABS(Z)
```

This will produce either -1 or 1 in parentheses as a first factor. We include the parentheses since mathematically, neither -1, nor 1, is a prime. By ending instruction 115 with a comma, instead of a semicolon, we also provided extra space between the unit and the prime factors.

As long as we are improving our program, we might as well make full use of the TRS-80's arithmetic ability by requesting it to use DOUBLE PRECISION arithmetic, which permits us to work with numbers as large as 16 digits, instead of 6. We can do this by adding:

```
10 DEFDBL Z, N, F
```

However, if we do this, we will be using the much slower double precision arithmetic, even when our number to be factored has 6 or fewer digits. To avoid this, insert:

```
10 DEFDBL Z
106 IF Z > 999999 THEN DEFDBL N, F
```

Many programmers would prefer to print a blank line or two and loop the program back to the beginning rather than stopping it once the number is factored. This is easily done by changing instruction 170 to

```
170 IF N=1 THEN 90 ELSE 140
```

and adding:

```

90 PRINT : PRINT
300 GOTO 90

```

Our revised program now reads:

```

10 DEFDBL Z
90 PRINT : PRINT
100 INPUT "TYPE NUMBER TO BE FACTORED "; Z
105 IF Z <> INT(Z) THEN PRINT "PLEASE USE INTEGER VALUES" : GOTO 100
106 IF Z > 999999 THEN DEFDBL N, F
110 PRINT " THE PRIME FACTORS OF "; Z ; "ARE ";
115 PRINT "(" ; SGN(Z); ")",
120 N = ABS(Z)
125 IF N/2 <> INT(N/2) THEN 135
130 PRINT "2";
132 N = N/2
134 GOTO 125
135 F = 3
140 IF N/F <> INT(N/F) THEN 200
150 PRINT F;
160 N = N/F
170 IF N = 1 THEN 90 ELSE 140
200 F = F + 2
210 IF F <= N THEN 140
300 GOTO 90

```

Although our new program takes more lines to write than the original program, actually it produces results in less than half the time, and works for many additional input values.

There are still three important cases to be considered. What happens if  $Z = 0$ ? Try it and see. If the output does not satisfy you, change the program further by adding:

```
IF Z = 0 THEN PRINT " some appropriate message."
```

Mathematically oriented readers may prefer to make the program even faster by using only prime values for  $F$  instead of 2, 3, 5, 7, 9, 11, 13, 15, ...Z. This will indeed save considerable time on factoring larger numbers, but requires that a list of prime numbers be stored or generated. We leave this as an exercise for those desiring additional speed. The critical statements are

```

READ F
DATA 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
DATA 53, 59, 61, 67, ..., etc.
DATA

```

where the DATA statements contain a list of all primes that are  $\leq \sqrt{9999999999999999}$  or  $< 100000000$  if 16-digit arithmetic is being used.

The point of this exercise is not to illustrate how to write a factoring program, but rather to show you that it is very easy to improve most programs, including commercial programs written in books (even this one). Just because a program "works" does not mean it is a good, or even an acceptable, program. The critical questions are:

1. Does it do the job for which it was designed? (Does it work?)  
If not, it is useless.
2. Can I improve it noticeably without much effort? If so, improve it and test the new program.
3. If it requires much effort to improve the program, will the results be worth the effort? (If the program will seldom be used again, and if it does not take an unreasonable amount of computer time as it is, it probably is not worth the effort to improve it unless you want to use it to brag a bit.)
4. You should always ask yourself if the method (algorithm) used by the original programmer is the best one you can think of.

Experienced programmers have a saying that contains much truth.

Only after you have written, debugged, and tested a program, do you understand it well enough to program it. If it is a major program, after you get the program all up and running smoothly, scrap the entire program and THINK carefully about it. Then start over from scratch and write a better program.



MORE MICRORESEARCH PROBLEMS

We examine another example in detail before trying new micro-Research Problems. We consider the four-digit case first, then after that is investigated, ask you to expand it in micro-research Problem 18, as suggested in the STEPS FOR COMPUTER ASSISTED PROBLEM SOLVING.

EXAMPLE 17-2

Let ABCD be a 4-digit number such as 4721

```
100 LET X = A
   A = ABS |A-B|
   B = ABS |B-C|
   C = ABS |C-D|
   D = ABS |D-X|
   PRINT A ; B ; C ; D
   IF A+B+C+D <> 0 THEN 100 ELSE STOP
```

Your first problem is to write a BASIC program that will show that each 4-digit starting-value number will converge to 0000 in 9 or fewer steps.

Write your own program before continuing.

EXAMPLE

```
ABCD = 4721
X = 4
A = |4-7| = 3
B = |7-2| = 5
C = |2-1| = 1
D = |1-4| = 3
```

```
NEW ABCD = 3513
X = 3
A = |3-5| = 2
B = |5-1| = 4
C = |1-3| = 2
D = |3-3| = 0
```

```
NEW ABCD = 2420
X = 2
A = |2-4| = 2
B = |4-2| = 2
C = |2-0| = 2
D = |0-2| = 2
```

```
NEW ABCD = 2222
X = 2
A = |2-2| = 0
B = |2-2| = 0
C = |2-2| = 0
D = |2-2| = 0
NEW ABCD = 0000
STOP
```

There are several ways this could be done. One of the more obvious is:

```

210 FOR M = 1 TO 9
220   FOR H = 0 TO 9
225     PRINT "WORKING ON M, H = "; M; H
230     FOR T = 0 TO 9
240       FOR U = 0 TO 9
250         A = M
260         B = H
270         C = T
280         D = U
290         PRINT "STARTING WITH A, B, C, D = " A; B; C; D
300         N = 0
310         X = A
320         N = N+1
330         A = ABS(A-B)
340         B = ABS(B-C)
350         C = ABS(C-D)
360         D = ABS(D-X)
370         PRINT A; B; C; D
380         IF A + B + C + D > 0 THEN 310
390         PRINT M;H;T;U; "TAKES ";N; "STEPS TO CONVERGE TO 0000."
400       NEXT U
410     NEXT T
420   NEXT H
430 NEXT M
500 PRINT
510 PRINT "FINISHED WITH 4-DIGIT TESTS."

```

This program runs through all of the four-digit starting-values, printing out new values as they are developed, and printing out the number N of steps each starting-value requires to converge to 0000 before going to a new starting-value whenever  $ABCD = 0000$ .

Put it on your computer and run it. It is interesting to watch, but it sure takes a long time to test 9000 cases!

Of course, you don't really need to watch, since, if a given starting value ever fails to produce 0000, the program will continue trying, and the change in pattern will be clear.

However, if we are not going to watch the program, it can be speeded up a lot by eliminating statements 290 and 370. Try it and see. Be sure you also recognize how to tell if this revised program ever finds a starting value which does not converge to 0000.

It is still a long program, and besides, it doesn't really do what we set out to do (to show that every 4-digit starting-value converges to

~~0000~~ in nine or fewer steps).

Hmm...let's store the largest N-value in another variable called L.  
(Why not in B for biggest?) We can do that by inserting:

```

205 L = 0
385 IF L >= N THEN 400
395 L = N
515 PRINT "LARGEST N VALUE IS " ; L

```

Let's see how that works.

```

205 L = 0
210 FOR M = 1 TO 9
220   FOR H = 0 TO 9
225     PRINT "WORKING ON M, H = "; M; H
230     FOR T = 0 TO 9
240       FOR U = 0 TO 9
250         A = M
260         B = H
270         C = T
280         D = U
300         N = 0
310         X = A
320         N = N+1
330         A = ABS(A-B)
340         B = ABS(B-C)
350         C = ABS(C-D)
360         D = ABS(D-X)
370         PRINT A;B;C;D;
380         IF A + B + C + D > 0 THEN 310
385         IF L >= N THEN 400
390         PRINT M;H;T;U; "TAKES ";N; "STEPS TO CONVERGE TO 0000."
395         L = N
400       NEXT U
410     NEXT T
420   NEXT H
430 NEXT M
500 PRINT
510 PRINT "FINISHED WITH 4-DIGIT TESTS."
515 PRINT "LARGEST N VALUE IS " ; L

```

When we ran the above program, we discovered  $L = 8$ . You can modify the program to determine all 4-digit starting-values that require 8 steps to converge to ~~0000~~.

microRESEARCH PROBLEMS

(continued from Lesson 5)

18. a. Extend Example 17-2 to 5-digit starting-values A B C D E

```

A = X
A = ABS(A-B)
B = ABS(B-C)
C = ABS(C-D)
D = ABS(D-E)
E = ABS(E-X)

```

etc. Let's print out each set (like we did in 290 and 370) to see what is going on at the early stages of our investigation.

It may come as a shock to you that not all values converge to 00000. Indeed, apparently quite a few converge to 000XX (where the digits XX are alike, but not necessarily 00), and then enter a long loop which eventually returns to 000XX.

- b. Modify the program or write a new one to investigate what really does happen to 5-digit starting-values.

- c. Extend your investigations to include 6, 7, and 8-digit starting-values and 2 and 3-digit starting-values, as well as the 4 and 5-digit values already investigated.

Would you like to conjecture that all starting-values converge to 00...XX, but that we can guarantee  $XX = 00$  only if the number of digits in the starting-value is a power of 2? Why, or why not?

19. Write a program that will shuffle 52 cards in the computer and deal out 4 bridge hands, and then bid one of the hands, with human input for bids on the other three hands.
20. Write a program to create Haiku poetry.
21. Write a program that will accept seven subscripted variables  $V(0)$ ,  $V(1)$ , ...,  $V(6)$  and will test to determine whether or not any of the variables contain equal values. If all seven variables are different, print "ALL VARIABLES ARE DISTINCT." Otherwise, print the subscripts and the common value of those variables containing the same value. If there is more than one set of subscripted variables with the same values, the program should also indicate the subscripts and common values of other sets.

```

V(0) = 4
V(1) = -3
V(2) = 1.765
V(3) = 4
V(4) = -2
V(5) = -3
V(6) = 4

```

The output should be

```

SUBSCRIPTS = 0 3 6  VALUE = 4
SUBSCRIPTS = 1 5  VALUE = -3

```

22. Write a program to accept one of the three symbols from the keyboard -  $\emptyset$  + . The program should then examine a string of DATA (from DATA statements or from tape) and if

- was depressed at the keyboard, display the number and sum of the negative DATA values.
- + was depressed at the keyboard, display the number and sum of the positive DATA values.
- $\emptyset$  was depressed at the keyboard, display the number of zeros in the DATA.

23. A theorem from number theory states that every integer  $N$  is expressible in the form

$$N = X^2 + Y^2 - Z^2 \quad \text{where } X, Y, \text{ and } Z \text{ are integers.}$$

- a. Write a program that will accept values of  $N$  between -100 and +100 and print out  $N$  (as the sum of two integer squares minus another integer square) before returning to accept another value  $N$ .
  - b. Enclose your program in a FOR...NEXT loop to test all  $-100 < N < 100$ .
  - c. Can you speed your program up so it could feasibly work for  $-5000 < N < 5000$ ?
24. Euclid's proof that there exist infinitely many prime integers uses the expression

$$N_k = 1 + (2*3*5*7*11*...*P_k)$$

where the product in the parentheses is the product of the first  $k$  positive primes.

Write a computer program to determine the first 20 values of  $N_k$ .

Extend your program to determine which of these  $N_k$  values are prime and what the factors of the non-prime  $N_k$  are. (Note: not all  $N_k$  are prime, but all of  $N_k$ 's prime factors are larger than  $P_k$  and less than  $(1 + \sqrt{N_k})$ ).

25. Knowing that our calendar repeats on a 400-year cycle, determine on which day of the week the 13th day of the month is most apt to occur. (Is it really Friday?)

26. Write a program to make a table showing the values of

N	$N^2$	R
---	-------	---

(Where R is the remainder when  $N^2$  is divided by 12.)

for  $N = 1, 2, 3, \dots, 20$ . Then jump to a conclusion (call it a conjecture, if anyone asks you) and then either prove or disprove your conjecture.

27. Determine the sum of the first  $k$  terms in the series

$$1 + 22 + 333 + 4444 + \dots$$

For a solution, consult page 807 (E 1405), American Mathematical Monthly Vol 67 (1960).

28. Determine all of the 4-digit integers which are perfect squares and in which all four of the digits are even. Your final program should produce the results in less than 3 seconds of TRS-80 Level II BASIC RUN time. If it takes much more than that, please do some additional analysis and try again.
29. Devise a program that will draw a dart target on the TRS-80 screen (squares are much easier to draw than circles), with numbers shown in each area (including negative numbers outside the target if you wish). Then permit a player to toss 5 darts by typing coordinates X,Y of a point. Add a little "wobble" to the point chosen by using  $X = X + (\text{RND}(9) - 5)$ ;  $Y = Y + (\text{RND}(5) - 3)$  or some such device before displaying SET(X,Y) as a (first blinking, then permanent) pixel on the screen. Your program should also keep a running total of the number of points the player made.

After the program is up and running--rewrite it so that the center of the target is not at the center of the screen. Also, permit the gamemaster to assign the values to each target area before the game starts.

30. The reverse of an integer (whole number) is the number obtained by writing its digits in reverse order—that is, the reverse of  $N=1089$  is  $R=9801$  and the reverse of  $N=13$  is  $R=31$ .

a) Your project is to find values of  $N$  such that  $N$  does not equal its reverse,  $N \neq R$ , but the product  $N \cdot R$  is a perfect square. Since  $1089 \cdot 9801 = 10673289 = (3267)^2$  such numbers clearly exist. How many can you find?

31. The numbers 12 and 21 are reverses of one another, as are 144 and 441, since they contain the same digits, but in reverse order. Interestingly,  $12^2 = 144$  and  $21^2 = 441$ . Find other pairs  $N$  and  $N'$  such that  $N$  and  $N'$  are reverses with  $N \neq N'$ , and such that  $N^2$  and  $(N')^2$  are also reverses.

32. You borrow an amount  $P$  dollars at  $I$  percent per annum interest rate on the unpaid balance. You agree to pay off  $M$  dollars per month. Write a program that will print out a table showing at the end of each month:

Amount of interest paid that month	Amount of principal paid that month	Remaining principal	Total Interest paid to date	Total paid to date
---------------------------------------	--	------------------------	--------------------------------	-----------------------------

Note that if in the last month (principal + interest) <  $M$  you should not have to pay the full  $\$M$ .

33. Lesson 4 contained a program designed to help you learn where the various pixel blocks were located on the screen. Extend the idea by first having the player type a difficulty number  $N$  where 1 is hard and 9 is easy, perhaps using

```
INPUT "PLEASE TYPE DIFFICULTY NUMBER BETWEEN 1 (HARD) AND 9 (EASY); N
```

Then give the player  $1+\text{RND}(N)$  chances to locate the lighted block with blinking pixel, before clearing the screen and choosing another randomly positioned lighted block.

34. Write a program to accept a positive integer  $K$  and then type out  $K$  primes of the form  $N^2 + 1$ .  
Example: If  $K = 3$   
the primes are 2, 17, 37  
each of which is prime and of the form  $(N^2 + 1)$ .

35. Write a program containing 20 or fewer BASIC instructions without using a PRINT" (PRINT *followed by quotes*) instruction to make a caricature of a face or person or animal.

36. Design a program which will serve as a timer for a series of 10-minute speeches. When the time keeper types RUN ENTER your program should display 10 MINUTES LEFT on the screen in normal size letters. The number of minutes should change every minute. When the speaker gets to 1 minute left the program should display some attention getting device on the screen and the switch to double size letters and display

0 MINUTES 59 SECONDS

58 SECONDS

etc. until 0 SECONDS is reached, whereupon the screen should go wild (possibly including an audio blast if you are equipped for audio output).

37. Prove or disprove that, for N a positive integer,

$$N^4 + 2N^3 + 2N^2 + 1 \text{ is}$$

never a perfect square. (Don't expect the computer to do this for you.)

38. Your local Utopia Society decides to replace their weekly Bingo session with a Numbers Betting Game to be available in gas stations and grocery stores. People can place bets on any three digit number from 000 to 999. The winning number is the three digits in the thousands places of the number of shares sold on the New York Stock Exchange that day. The payoff is 500 times the amount bet for bets up to \$20. For bets above \$20 the payoff is \$10,000 plus 100 times the amount bet.

Your first task is to write a program to simulate the betting system and come up with estimates of likely profits or losses. You should also discover (with or without using a computer) what amount of bet will produce the greatest payoff for the bettor. If all bettors wagered this amount (best possible for them, worst for the Utopians), how much would you expect the Utopia Society to make or lose on each approximately \$10,000 that was bet? Assume that 10¢ of each dollar bet goes to the person taking the bet, and 15¢ of each dollar bet is other overhead.

39. Compute and print the number  $2^{75}$  exactly.



40. Find all positive integers X,Y,Z such that

$$X^2 + Y^2 = Z^2 \text{ with } Z < 1000.$$

Compare the speed of your program with the speed of programs written by your friends.

41. a) Two rods of equal length are divided into 250 and 243 equal parts respectively. Their ends are coincident. Find the divisions which are nearest together.

b) Same problem but divide the two equal rods into M and N equal parts. Write a program to accept positive integer input M and N and display the two values which are nearest together.

42. Consider the recursive relation defined by:

Let  $B$  = The biggest number that can be made by rearranging the digits of  $N_k$ .

$S$  = The smallest number that can be made by rearranging the digits of  $N_k$ .

$$N_{k+1} = B - S$$

- a) If  $N_0$  is a four-digit starting value, one of two things happen:
- i) If all the digits of  $N_0$  are identical, the  $N_{k+1} = 0$  for all  $k \geq 0$ .
  - ii) If  $N_0$  has at least two distinct digits, then the relation converges to  $N_{k+1} = 6174$  which then repeats forever. Investigate this phenomenon.
- b) If  $N_0$  is a six-digit integer, there are 384 non-zero possible values of  $N_1$ , some of which lead to the repeater 631764, but some of which do not. Investigate other starting values  $N_0$  having 6 digits.
- c) Investigate the  $N_{k+1} = \text{BIG} - \text{SMALL}$  phenomenon for starting values of other sizes.
43. It is well-known that the reciprocals of integers form repeating decimals of period  $p$  (we say,  $p = 0$  if the expansion terminates as on  $1/2 = .500$ ;  $p = 1$  for  $1/3 = .33$ ; etc.) Our problem is to create a table which will give the smallest positive integer  $N(K)$  such that its reciprocal has a repeating decimal expansion of length  $K$ .

44. If  $N_0$  is a positive integer, it can be shown that the recursive relation  $N_{k+1} =$  (The sum of the squares of the digits of  $N_k$ ) will either converge to 1 or will reduce to a self-repeating cycle 37, 58, 89, 145, 42, 20, 4, 16, 37, ...
- a) Do so. Can you prove that every positive starting value  $N_0$  will eventually either converge to 0,0,0, or to the cycle 20, 4, 16, 37, ...
- b) The related problem for the sum of other powers of the digits also merits investigation. Do so.
45. It is a common practice in mapping to approximate the area of a lake or island or other irregular shape by using an irregular polygon, whose area is easier to compute than is the area inside a curved boundary.

Write a program into which the number  $N$  of vertices and the  $X$  and  $Y$  coordinates of the vertices  $(X_i, Y_i)$  of the polygon in consecutive order may be input and the program will then compute the polygonal area. It will be necessary to either find or devise a suitable formula. Remember this polygon is not a regular polygon since its sides may be of unequal length.

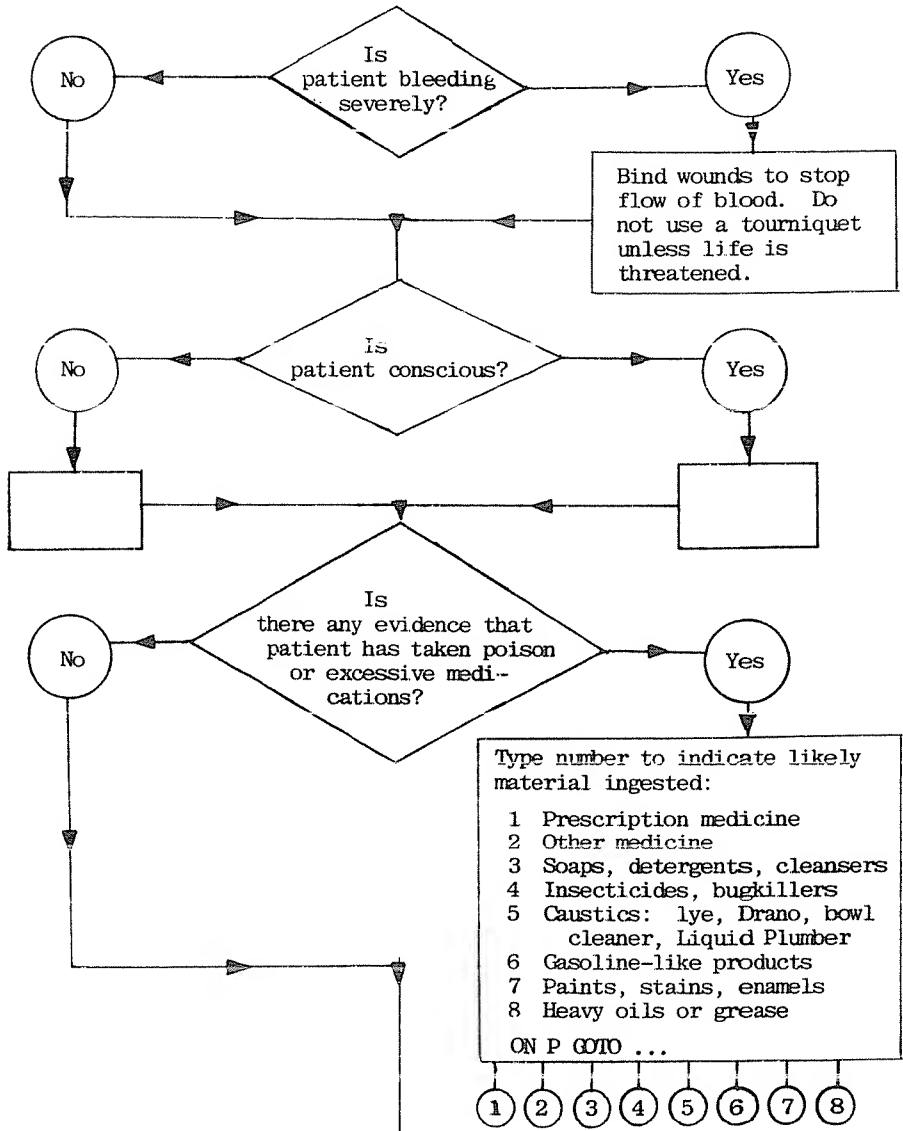
46. Given two values  $A$  and  $B$ , determine values  $X$ ,  $Y$ , and  $Z$  such that

$$\frac{A}{X} = \frac{X}{Y} = \frac{Y}{Z} = \frac{Z}{B}$$

What will your program do if  $A$  and  $B$  have different signs?

47. Liouville discovered an interesting procedure for producing sets of positive integers with the property that the sum of their cubes is equal to the square of their sum. One such set is  $1^3 + 2^3 + 3^3 + 4^3 = (1 + 2 + 3 + 4)^2$ . A bit of induction will show that for all  $N$ ,  $1^3 + 2^3 + 3^3 + \dots + N^3 = (1+2+3+\dots+N)^2$ . However, it is not known whether or not Liouville's method will produce all such sets. Write a program to produce such sets in a given range. Investigate the problem.
48. Write a program that will compute how much an investment of  $P$  dollars will be worth  $Y$  years from now if it is invested at  $I$  percent per annum interest, compounded  $N$  times per year. A useful formula is:
- $$V = P(1 + I/(100*N))^{N*Y} \quad (\text{Where } V \text{ is the value after } Y \text{ years.})$$

49. Design a program to coach emergency first-aid treatment at the "Boy Scout" level. The program should ask questions answerable by YES or NO and prescribe action based on the response.



50. Either find a ten-digit positive integer  $N$  with each digit different, and such that  $N$  is also a perfect 10th power of an integer, or show that no such  $N$  exists.
51. Consider a large lattice of points (100 x 100 would be large in this case and a big sheet of graph paper will do nicely) with equal spacing in the  $x$  and  $y$  directions. Our real problem is to place as many points as possible on the lattice subject to the restriction that no subset of three points shall form a right triangle. Before tackling the real problem, consider the much simpler problem of a program to find and print out 6 or 8 patterns of points on the lattice which have the property that no three points form a right triangle and that the set is maximum in the sense that if one more point were added to the given arrangement, a right triangle would be formed. Possibly placing points down the diagonal has this property. Then try to answer (with proofs) the questions:
- What is the largest number of points which may be so placed?
  - What is the smallest number of points in a maximum set?
  - Restate the problem so that using every point of two adjacent sides of the square excepting the corner point is not a solution and solve the new problem.
52. Write a program to sort a set of 25 names alphabetically. Ponder this one. Sorting represents one of the most time-consuming uses of modern computers. Consider sorting and entering the approximately 100,000 grades received by 20,000 students each term at a major university.
53. J. Maxfield has shown (Math Magazine, Vol. 43, #2 March, 1970, pp.64-67) that given a sequence of digits  $K = d_1 d_2 d_3 \dots d_n$ , there exists an integer  $N$  such that  $N!$  begins with the sequence of digits  $K$ . Write a program to create a table such that the smallest such positive integer  $N(K)$  is determined for each  $K \leq 100$ . You may also wish to print out the floating point value of  $N!$  if it will be available at no additional cost. Eventually, we would like a formula for  $N(K)$  = smallest positive integer  $N$  such that  $N!$  begins with the sequence of digits  $K$ , but that may be too much to hope for. Your table might well resemble the one below:

$K$	$N(K)$	$N!$
1	1	1
2	2	2
3	9	362880
4	8	40320
5	7	5040
6	3	6
7	6	720
:	:	:
:	:	:

54. Mathematicians have proved that there exist arbitrarily long strings of consecutive positive integers such that each of the consecutive integers has as a factor, a perfect square greater than one (probably different squares for different integers). This may be restated as follows:

Given a positive integer  $N$ , there exist strings of  $N$  consecutive positive integers each of which contains a factor which is a perfect square greater than one.

If  $N = 3$                       48, 49, 50 contain  $4^2$ ,  $7^2$ ,  $5^2$  as factors.  
                                     98, 99, 100 contain  $7^2$ ,  $3^2$ ,  $10^2$  as factors.

If  $N = 4$                       242, 243, 244, 245 contain  $11^2$ ,  $3^2$ ,  $2^2$ ,  $7^2$  factors.

If  $N = 7$                       217070, 217071, 217072, 217073, 217074, 217075, 217076  
 contain as  
 factors                       $7^2$          $3^2$          $2^2$          $113^2$          $11^2$          $5^2$          $4^2$

Your problem is to find the first (i.e., involving smallest integers) string of  $N$  consecutive integers each of which contains a square  $>1$  as a factor for  $N = 2, 3, \dots, 10$ .

55. This one is tough! Mathematicians have shown a similar theorem to that given in Problem 54 holds with "perfect square" replaced by "perfect cube" or even "perfect  $K$ th power". Investigate this for  $K = 3, 4, 5$ , and  $N = 2, 3, \dots, 10$ . (Or even  $N = 2, 3, 4, 5, 6$ , for starters.)

Examples:	3rd Powers ( $K = 3$ )				4th Powers ( $K = 4$ )			
$N=2$	80	81			80	81		
	8	27			16	81		
$N=3$	1375	1376	1377		33614	33615	33616	
	125	8	27		2401	81	16	
$N=4$	22624	22625	22626	22627				
	8	125	27	1331				

56. Find a set of distinct positive integers, each of which is less than 100, such that it satisfies both of the following conditions.
- No combination (or subset) of them added together will total 100.
  - The set contains as many distinct, positive integers as possible.

Hint: You may be able to solve this without using a computer.

57. The square of the number 363636364 has the unusual property ( in base 10) that the second half of the square is an exact duplicate of the digits in the first half

$$(363636364)^2 = 13223140496,13223140496$$

as shown by the central comma above.

Such periodic squares also exist in bases other than 10. For example, in base 4,

$$(21213_4)^2 = 11301,11301_4 = 378225_{10} = (615_{10})^2$$

Your problem is to write a program that will examine perfect squares, expressed in some base  $B > 1$ , (use base  $B = 10$ , if you wish) and find perfect squares whose expression in base  $B$  is periodic (period 2 or more). See Mathematics Magazine, March 1975, p. 97.

58. An integer (whole number) greater than 1 is said to be prime if its only positive factors are itself and one. There are many  $4 \times 4$  arrays of digits such that each row and each column is a 4-digit prime.

Example:    6   1   8   9            Where    6189    6359  
              3   0   2   3                    3023    1021  
              5   2   7   3                    5273    8273  
              9   1   3   7                    9137    9337   are each prime.

- a) Find some additional such arrays. Then, hunt for arrays whose main (upper left to lower right) diagonal also contains a prime.  
 b) What happens if you try to find  $5 \times 5$  arrays containing 5-digit primes in each row, each column and main diagonal?

59. Write a program to plot a graph     $Y = f(X)$ .

Your program may request the user to put the function in a certain statement. It should ask the user for what range of  $X$ -values he wishes the graph plotted and then compute (or ask the user) for the largest and smallest  $Y$ -value that will be used. It should then scale the function suitably for display on the CRT screen. Your program should also display for the user (perhaps using `PRINT @ 960, " "`) the  $X$ -range and the  $Y$ -range covered by the graph.

60. Find the smallest positive integer  $M$  such that

$$N = 7 \cdot 10^M + 1 \quad \text{is not prime.}$$

61. Find and print all the five-digit positive integers which contain exactly the same digits as their trebles. The leading digit must not be zero.

Example:  $10035 * 3 = 30105$

Please also record the computer time used on your final run. Compare with that of other TRS-80 users you know.

62. Many people from Archbishop John Tillotson (c. 1650) through Kurt Vonnegut (1950) and beyond have toyed with the idea that if enough monkeys were permitted to pound typewriters for long enough, all the great classics of literature would eventually be reproduced. We can easily improve the chances of something legible being produced by picking keys in the approximate order of their frequency in normal English rather than at random. Write a program that will print out the following symbols, with the given approximate frequency per 1000. To simplify the problem we restrict our selection of punctuation to "space", and have omitted V, K, Q, X, J, and Z, which have very low frequency in English.

KEY	space	E	T	A	O	N	I	R	S	H
FREQUENCY/1000	223	100	77	62	62	54	54	54	46	38
KEY	D	L	F	C	M	U	G	Y	P	B
FREQUENCY/1000	31	23	23	23	23	15	15	15	15	15

You may be able to further improve the likelihood of obtaining at least occasional words by including digraphs (two-letter groups) as well as single letters. English digraph frequencies are approximately

DIGRAPH	TH	HE	ER	AN	IN	ON	RE	AT	ED	ST	ND	ES
FREQUENCY/1000	38	31	18	17	17	16	14	14	13	12	12	12

Additional frequency data for English, French, Spanish, German, and Italian are available in Sophisticated Ciphers, available from Mu Alpha Theta Math Club, Room 423, 601 Elm street, Norman, OK, 73019, or at your library.

63. Find all n-digit positive integers in which the n-digit number is an exact n-th power of an integer.

Examples:  $81 = 9^2$        $125 = 5^3$

64. Two positive integers are amicable (friendly) if the proper (including one, but not including the number itself) divisors of each add up to the other. 220 has divisors 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110, whose sum is 284. 284 has divisors 1, 2, 4, 71, 142, whose sum is 220, thus 220 and 284 are said to be amicable. Some other amicable pairs are (17296, 18416) and (1184, 1210). Write a program that will find all amicable pairs between two given bounds S and B and which will also keep track of any number which has a divisor sum greater than B, so these can be used in a future run with (new S) = (old B) and a larger B. Your final program should also account for any "carry over" values from previous runs.
65. Amicable pairs (problem 64) have been generalized to amicable chains (sociable chains) in which the sum of the proper divisors of each number in the chain is the next number and the last member of the chain has the first member as the sum of its divisors. The only chains generally known are the two four-link chains (found in 1965):  
 (2115324, 3317740, 3649556, 2797612) and  
 (126460, 1547860, 1727636, 1305184),  
 the five-link chain:  
 (12496, 14288, 15472, 14536, 14264),  
 and a stupendous 28-link chain which contains the number 14316.  
 However, there are others.
- a) Write a program to investigate amicable chains containing 50 or fewer links.
- b) What happens to chains that do not loop back to the starting value?
66. Write a program that will match up 50 men and 50 women as dance partners. Your first attempt can use numbers, if you wish, but the final result should be two lists; one with the men's names in alphabetical order and showing the randomly matched partner's name, the other with the women's names in alphabetical order and showing the name of each woman's partner (hopefully, the same matching as above). Each person should be matched with one and only one partner and that partner should be of the opposite sex.
67. a) Write a computer program to accept three values M, D, Y and determine whether or not they are acceptable values for Month, Day, Year, in that order.
- b) Extend the above program to accept two sets of three numbers and if both are acceptable as Month, Day, Year values, determine the number of days between them, including both dates in your count. Example: 2, 20, 1978 to 2, 22, 1978 is 3 days.



68. A computer is apt to obtain different values for  $F(N) = 1*2*3*\dots(N-1)*N$  and for  $G(N) = N*(N-1)*\dots 3*2*1$ . Furthermore, the difference is apt to be non-trivial. For  $N = 30$ , the difference can be greater than a quintillion (and that's more than the number of cents in the national debt). Investigate this phenomenon, using a program of your own devising, or using:

```

1000 FOR N = 10 TO 50
1100   F = 1
1200   G = N
1300   FOR K = 2 TO N
1400     F = F*K
1500     G = G*(N-K+1)
1600   NEXT K
1700   PRINT N, F, G, ABS(F-G)
1800 NEXT N

```

69. Mathematicians have shown that given a sequence of digits  $S = d_1d_2d_3\dots d_n$  there exists an integer  $N$  such that  $2^N$  begins with the sequence  $S$ . (As a matter of fact, for 2 we may substitute any positive integer which is not a power of 10, i.e., not  $10^0=1$ ,  $10^1=10$ ,  $10^2=100$ , etc.)

Your problem is to produce a table giving  $S$ ,  $N$ ,  $2^N$  for  $S = 1$  to 100 such that  $N(S)$  is the smallest positive integer such that  $2^{N(S)}$  begins with  $S$ . I think the first few values are:

S	Smallest $N > 0$	$2^N$
1	4	16
2	1	2
3	5	32
4	2	4
5	9	512
6	6	64
7	46	70368744177664
8	3	8
9	53	etc.
10	10	
11	50	

70. Problem 11 (Lesson 5) asks you to generate and print out in increasing order of size, the positive integers of the form  $N = 2^k \cdot 3^m$  where  $k \geq 0$  and  $m \geq 0$ . Generalize this to integers of the form  $A^k \cdot B^m$  where  $A$  and  $B$  are read using an INPUT statement. Note that  $A$  and  $B$  need not be prime. Your program should work if  $A = 6$  and  $B = 10$ .

71. Use an array containing the proper fractions in lowest terms to establish a "position number" for each entry. In the array each entry in row  $k$  shall have  $(k+1)$  as denominator and the numerators in a given row shall be in increasing order.

1/2					
1/3	2/3				
1/4	3/4				
1/5	2/5	3/5	4/5		
1/6	5/6				
1/7	2/7	3/7	4/7	5/7	6/7
1/8	3/8	5/8	7/8		
1/9	etc.				
1/10	etc.				
.					
.					
.					

Since only those proper fractions in lowest terms are included, the rows are not of easily predictable length.

Establish a position number  $N(\alpha)$  for each fraction  $\alpha$  in the array by simply counting the entries, row after row, from left to right until  $\alpha$  is reached.

Thus

$N(1/2) = 1$
$N(1/3) = 2$
$N(1/4) = 4$
$N(2/3) = 3$
$N(3/4) = 5$

and so forth. In this fashion one establishes that

$N(1/13) = 46$	
$N(15/16) = 79$	
$N(1/100) = 3004$	
$N(3/500) = 75917$	
$N(7/1240) = 467048$	(or do they?)

The challenge is to determine formulae or an algorithm that will enable you to do either (or both) of the following

1. Given a proper fraction  $\alpha$  in lowest terms, determine  $N(\alpha)$ .
2. Given an integer  $K$ , determine the corresponding proper fraction  $\alpha$  such that  $K = N(\alpha)$ . Use a computer to help if you can.

72. Program a game of your own choice on the computer.

73. The problem of placing eight queens on an 8x8 chess board so that no queen can attack any other queen (queens attack any piece they can reach by moving in a straight line horizontally, vertically, or on either diagonal) is an old one. You may have considered the problem before. You will find it discussed on p. 161-2 of W. W. Rouse Ball's interesting book, Mathematical Recreations & Essays or in Chapter 10 of M. Kraitchik's admirable book, Mathematical Recreations, both of which should be available at your local library.

You may write a program to solve the problem if you wish, but the problem proposed here is to examine the following program and see if you can figure out how it works and why it produces all 92 possible solutions to the eight queens program.

```

2 CLS
4 PRINT "      EIGHT QUEENS PROBLEM   (RUNNING TIME ABOUT 40 MINUTES)"
6 PRINT
8 PRINT "          1   5   8   6   3   7   2   4  MEANS"
10 PRINT " THE QUEEN IN THE FIRST COLUMN IS ON ROW 1;"
12 PRINT "  THE QUEEN IN THE SECOND COLUMN IS ON ROW 5;"
14 PRINT "   THE QUEEN IN THE THIRD COLUMN IS ON ROW 8;ETC."
16 PRINT
20 DIM R(8)
50 N = 0
100 C=0
200 C=C+1
300 R(C)=0
400 IF R(C) < 8 THEN 600
500 C=C-1
510 IF C = 0 THEN PRINT"END OF SEARCH":STOP
520 GOTO 400
600 R(C) = R(C) + 1
605 IF C < 2 THEN 660
610 FOR C1 = 1 TO C-1
620   IF R(C)=R(C1) OR C+R(C)=C1+R(C1) OR R(C)-C=R(C1)-C1 THEN 400
650 NEXT C1
660 IF C < 8 THEN 200
690 N=N+1:PRINT"SOLUTION#";N;"=",
700 FOR C1=1 TO 8
710   PRINT R(C1);" ";
720 NEXT C1
730 PRINT
740 GOTO 500

```

74. If you are seriously interested in more advanced computing, investigate one of the following languages for your TRS-80.

"C" or "tinyC"

"PASCAL"

"Z-80" chip language (enter from BASIC II throughUSR( ) function)

PILOT (a language designed for Computer Assisted Instruction)

75. Study a program written by someone else, and then improve it.

## A FINAL WORD from your authors

### Good Programming Practices

There is no such thing as the correct programming style. Entire books have been written on the subject, but even the most skillful practitioners of the programming art disagree on what is best. However, there are several common sense principles that merit consideration. Even these are not immutable rules, but it is prudent to follow the suggestions unless you have a good reason for not doing so on a given program. Here goes:

1. Your program should, in general, read from top to bottom in a logical fashion, except for subroutines which usually are listed after the completed program. Block the various fragments into different 100's in statement numbers. Don't feel compelled to use every tenth statement number.
2. Use occasional REM statements to explain what is happening. If you must use a GOTO statement in the heart of your program, include a REM statement explaining why.  
Your programs will have a neater appearance, and be easier to debug if you place five blank spaces between REM and the start of your remark. Try it, you'll like it.
3. Whenever you use a FOR-NEXT loop, indent the instructions between FOR and NEXT three spaces. If you use nested FOR-NEXT loops, stair-step them. It makes programs much easier to understand.
4. Use the faster forms whenever feasible.  
(See Lesson 11.)
5. Get to know your computer. Become thoroughly familiar with the quirks and excentricities of your computer and of the BASIC (or other) language you are using.

Above all:

Don't let these or any other rules spoil your fun--but do remember that well-written, easy-to-follow programs are no harder to think or write than sloppy kluges and are much more profitable in use and maintenance. Think of a computer program as a living, growing, ever-changing thing, not as a finished, static object.

H A V E \_ \_ F U N

*Josephine Andree*  
*Richard V. Andree, 1980*

## A

ABS(*exp*), 186  
 Algorithm, 95  
 AND, 179  
 Arithmetic Functions, 186  
     ABS(*exp*), ATN(*exp*), CBEL(*exp*),  
     CINT(*exp*), COS(*exp*), CSNG(*exp*),  
     EXP(*exp*), FIX(*exp*), INT(*exp*),  
     LOG(*exp*), SIN(*exp*), SGN(*exp*),  
     SQR(*exp*), TAN(*exp*)  
 Arrays, 147  
 "Art", Random, 66  
 ASC(*string*), 173  
 Asteroids, Game Program, 119  
 ATN(*exp*), 186  
 AUTO *mm*, *nn*, 184

## B

Backspace Key, 5, 26  
 Bar Graph, 54, 55  
 Biological Simulations, 131, 134  
 Bounce, Keyboard, 11, 108  
 Branching, 23  
**BREAK** Key, 2, 5, 7, 11, 26

## C

Campus Planning Program, 56  
 Cassette Program Loading, 105  
 Cautions, 150  
 CBEL(*exp*), 186  
 Challenge Problems, 32, 45, 46  
 Changes, 125  
 Changing Instructions, 11  
 CHR\$  
     ( ), 61, 62, 63, 165, 169, 173  
     ( ), PRINT, 166  
     ( ), Effects of, 61-63  
 CINT(*exp*), 186  
 Cipher Program, 175  
**CLEAR** Key, 2, 5, 12, 26  
 CLOAD, 105  
 CLS, Clear Screen, 22  
 Code Message Program, 175  
 Comma, 9, 12  
 Commands in Your Programs,  
     Keyboard, 157  
 Computer-Assisted Problem Solving,  
     Steps For, 80

Concatenate Strings, 174  
 Conference Proceedings, 181  
 Continuous Function, 39  
 Correction, Error, 40  
 COS(*exp*), 186  
 Counselor, Robot, 176  
 CSAVE, Saving a Program On Tape, 107  
 CSNG(*exp*), 186

## D

DEF, 143  
 DEFDBL, 86, 144  
 DEFINIT, Variables, 144  
 DELETE *mm-nn*, 184  
 Deleted, 124  
 Designators, Variable Type, 145  
 Dice Throwing Program, 130  
 Display, Truncated, 21  
 Double Letters, 108, 150  
 Double Precision Arithmetic, 86  
 Double Precision Variables, 143  
     In FOR...NEXT Loops, 151  
 Double Size Letters, 12, 16, 26

## E

EDIT, 123-129  
 Edit Mode Subcommands, 126  
 Editing, 40  
 Ehrenfest Model, Three Molecules,  
     "Guppy" Problems, 135  
**ENTER**  
     Key, 2, 5, 26  
     LIST, 8, 26  
     NEW, 26  
     RUN, 26  
 E+08, 12  
 Equation Solver, 34-39  
 Erase, 5, 26  
 ERROR  
     Correction, 40  
     Messages, 155, 156  
 EXP(*exp*), 186  
 Extended Print Instructions, 159

## F

Factor Program, 191  
 Factorial, 44  
**FALSE**, 179  
 Fast N Such That  $N^2$  Ends in N, 92  
 Faster Forms, 154  
 Field Specifiers, PRINT USING, 161  
 Fix, Keyboard, 11  
 FIX(*exp*), 186  
 Floating-Point Variables, 12, 143  
   Double Precision, 86, 143  
 Flow Chart, 137, 138, 190  
 FOR K = 0 TO 20 STEP .5...NEXT K, 42  
 FOR Q = 1 TO 400: NEXT Q, 68  
 FOR...NEXT, 33  
 Forms, Faster, 154  
 FRE(*string*), 173  
 Free Improvements For Cassette  
   Use, 103  
 Functions, Continuous, 39  
 Functions, Graphing, 64

## G

Game Program, 121  
   Asteroids, 119  
   Guessing Game, 113  
   Hangman, 114  
   Pixel Hunt, 54  
   Reverse Memory Test, 117  
   Treasure Hunt, 112  
 GOSUB *nn*, 185  
 GOTO *nn*, 8  
 Graph, 168  
 Graph, Bar, 54, 55  
 Graphics, 165  
 Graphing, 163  
 Graphing Functions, 64, 67  
 "Guppy" Problem, Three Molecules,  
   Ehrenfest Mode, 135

## H

Hangman, Game Program, 114  
 Headings, 23, 25  
 Histogram, 55  
 Hunt, Pixel Hunt Game, 54

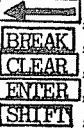
## I

Improving Programs, 110, 194  
 Information, Additional, 180  
 INKEY\$, 173, 175  
 Instructions: See Instruction Index

## J

Jargon Phrase Maker, 171  
 Journals, 181

## K

Key  
  
   BREAK, 3, 5, 26  
   CLEAR, 2, 7, 11, 26  
   ENTER, 2, 5, 12, 26  
   SHIFT, 2, 4, 5, 26  
   SHIFT, 1

Keyboard, 1  
 Keyboard Bounce, 11, 108  
 Keyboard Commands, 157  
 Keyboard Fix, 11  
 Keyboard Instructions, 18

## L

Large Numbers, 84  
 Last 3 Digits, 81  
 LEFT(*string*, *n*), 174  
 LEN(*string*), 173  
 LIST **ENTER**, 3, 8, 26, 184  
 LIST *mm-nn*, 184  
 LOG(*exp*), 186  
 Logical Operators, 178  
 "Loop, Tight", 60

## M

Malfunctions, 192  
 Map, Video Screen, 49, 52  
 MEMORY SIZE, 6  
 Message, 57  
 Micro Research Problems, 75, 97,  
   188, 198  
 MID\$(*string*, *p*, *n*), 174  
 Music, Sound and, 108

## N

NEW, 7, 11, 26  
 NOT, 179  
 Numbers, Random, RND( $\emptyset$ ), RND( $n$ ), 47  
 Numeric Character, 161

## O

ON N GOSUB  $mm$ ,  $nn$ ,  $pp$ ,  $qq$ , ..., 185  
 ON N GOTO  $a$ ,  $b$ ,  $c$ , ...,  $z$ , 68  
 ON N GOTO  $mm$ ,  $nn$ ,  $pp$ ,  $qq$ , ..., 185  
 Operators, Logical, 178  
 OR, 179

## P

PEEK, 187  
 Periodicals, 181  
 Pixel Hunt Game, 54  
 Pixels, 52, 166  
 POINT( $x$ ,  $y$ ), 53  
 POKE, 187  
 POKE Address, Pixel Number  
   Instruction, 167  
 Precision Variables, Double, 143  
 PRINT, 2  
   @, 48, 68  
   CHR\$( $n$ ), 68, 166  
   STRING\$, 163, 164  
   USING Field Specifiers, 161  
 Problem: N Such That  $N^2$  Ends in  
   N, 79  
 Problems, Challenge, 32, 45, 46  
 Proceedings, Conference, 181  
 Program,  
   Factor, 191  
   Game, 112, 113, 114, 117, 119, 121  
   Load From Cassette, 105

## Q

Question Mark ?, 41

## R

Radio Shack Computer Center, 180  
 RANDOM, 129, 187  
 Random "Art", 66  
 Random Numbers, 47, 129  
 READY, 6

Research (micro) Problems You May  
 Undertake, 97, 198

RESET( $x$ ,  $y$ ), 53  
 RESTORE, 185  
 RETURN, 185  
 Reverse Memory Test Game, 117  
 RIGHT\$( $string$ ,  $n$ ), 174  
 RND

(K), 129  
 ( $n$ ), 68  
 ( $\emptyset$ ), 68, 129  
 Robot Counselor, 176  
 RUN 3, 68  
 ENTER, 26  
    $mm$ , 184

## S

Saving Computer Time, 19, 89, 92,  
   154, 196ff  
 Saving Memory Spabe, 153  
 Saving Program On Tape, CSAVE, 107  
 Screen, Video Map, 49, 52  
 Scrolling, 51  
 Semicolon, 12  
 Sentence Generator, 170  
 SET( $x$ ,  $y$ ), 53  
 SGN( $exp$ ), 186  
 SHIFT, 1, 12, 26  
 Simulate, 139  
 Simulation, 129, 131  
 SIN( $exp$ ), 186  
 Snowflake, 69  
 Solver, Equation, 34-39  
 Sound and Music, 108  
 Space, Out of String, 150  
 Speeding Up Programs, 154  
 SQR( $exp$ ), 186  
 Static Electricity, 152  
 Statistics, 146  
 Steps For Computer-Assisted  
   Problem Solving, 80  
 Store the Words in DATA  
   Statements, 170  
 String Concatenation, 174  
 String Instructions, 173, 174  
 Strings, 169-179  
 STRING\$( $K$ , "character" or number),  
   163, 174  
 STR\$( $exp$ ), 174  
 Subcommands, Edit Mode, 126

Subscripted Variables, 146  
Symbol >-, 5, 12, 41  
SYSTEM, 184

## T

TAB(*exp*), PRINT, 160  
Tables, 19  
TAN(*exp*), 186  
Tape  
    Care, 104  
    Cassette, 103  
Three Molecules, Ehrenfest Model,  
    "Guppy" Problem, 135  
"Tight Loop", 60  
Time, 88  
Tips, 150  
TRACE, 185  
Treasure Hunt, Game Program, 112  
TROFF, 185  
TRON, 185  
TRUE, 179  
Truncated Display, 21  
Twelve Days of Christmas, 46, 75

## U

USR(*n*), 185

## V

VAL(*string*), 174  
Value, PRINT USING String, 161  
Variable  
    DEFDBL, 144  
    DEFINT, 144  
    Double Precision, 143  
    Floating-Point Single  
        Precision, 143  
    Integer, 144  
    String, 144  
    Subscripted, 146  
    Type Designators, 145  
    Types, 143  
Video Screen Map, 49, 52  
Volume, 105

## W

Wages, 45  
Wild Screen, 120  
Word Generator, 171







# Special Characters

- Abbreviation for :REM
- % Makes variable integer-precision.
- ! Makes variable single-precision.
- # Makes variable double-precision.
- \$ Makes variable string type.
- : Separates statements on the same line.
- ? Same as PRINT (but L? can't be substituted for LPRINT).
- PRINT punctuation: spaces over to the next 16-column PRINT zone.
- : PRINT punctuation: separates items in a PRINT list but does not add spaces when they are output.

# Error Messages

Abbreviation	Explanation
NF	NEXT without FOR
SN	Syntax error
RG	RETURN without GOSUB
OD	Out of data
FC	Illegal function call
OV	Overflow
OM	Out of memory
UL	Undefined line
BS	Subscript out of range
DD	Redimensioned array
0	Division by zero
ID	Illegal direct
TM	Type mismatch
OS	Out of string space
LS	String too long
ST	String formula too complex
CN	Can't continue
NR	No RESUME
RW	RESUME without error
UE	Undefined error
MO	Missing operand
FD	Bad file data
L3	Disk BASIC feature

# Edit Commands

**EDIT** Puts computer into edit mode for specified Edit Commands.  
 EDIT 100 EDIT.

- (A) Cancels changes and starts again.
- (C) Changes n characters.
- (D) Deletes n characters.
- (E) Ends editing and saves all changes.
- (H) Hacks line and inserts at end.
- (I) Inserts characters.
- (K)c Kills all characters up to nth occurrence of c.
- (L) Lists the line.
- (Q) Quits edit mode and cancels all changes.
- (S)c Searches for nth occurrence of c.
- (X) Extends line (inserts at end).
- (SHIFT) (I) Causes escape from command.
- (ENTER) Records all changes and exits edit mode.
- (SPACEBAR) Moves cursor n spaces to the right.
- (←) Moves cursor n spaces to the left.

# Control Keys

- (←) Cancels last character typed; moves cursor back one space.
- (SHIFT) (←) Erases current line.
- (BREAK) Interrupts anything in progress and returns to command level.
- (CLEAR) Clears the screen.
- (ENTER) Signifies end of current line.
- (SPACEBAR) Enters a space (blank) character and moves cursor one space forward.
- (→) Advances cursor to next tab position.
- (SHIFT) (→) Puts display in 32-character mode.
- (I) Line feed and carriage return.
- (SHIFT) (I) "Control" key—hold down these two and press any key A-Z for control A - control Z.
- (SHIFT) (I) (C) Copies the display contents to the printer.
- (SHIFT) (C) Causes currently executing program to pause (press any key to continue).